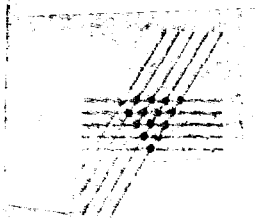
(2)

The Beckman Institute for Advanced Science and Technology



Permissive Planning: A Machine-learning Approach to Planning in Complex Real-world Domains

Scott William Bennett

University of Illinois

Artificial Intelligence
Technical Report UIUC-BI-AI-93-01

The University of Illinois at Urbana-Champaign

Permissive Planning: A Machine-learning Approach to Planning in Complex Real-world Domains

Scott William Bennett

University of Illinois

Artificial Intelligence
Technical Report UIUC-BI-AI-93-01

The Beckman Institute
University of Illinois
405 North Mathews Avenue
Urbana IL 61801
Tel: (217) 244-1983
Fax: (217) 333-1878
E-mail: scollins@cs.uiuc.edu

This paper was originally a thesis submitted in partial fulfillment
of the requirements for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the University of Illinois at Urbana-Champaign, 1993.

Technical report issued January, 1993

Beckman Institute Technical Reports are printed on recycled and recyclable paper.

PERMISSIVE PLANNING: A MACHINE-LEARNING APPROACH TO PLANNING IN COMPLEX REAL-WORLD DOMAINS

Scott William Bennett, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 1993
Gerald F. DeJong, Advisor

Classical planning techniques have some serious problems when employed in real-world domains. In classical planning, it is assumed we know the current state of the world and can project that state through a reasonably well-defined set of actions to yield a future state. However, perfect models of the world and of operators are not possible in most domains. Consequently, discrepancies occur between the projected future state and the observed future state. In these complex domains, the success of the plan can never be guaranteed. Furthermore, an important tradeoff exists between the time spent constructing a plan and its resulting chance of success. Several approaches to these problems have been investigated, including the use of decision-theoretic methods and the incorporation of reactivity into planners. We present a new technique called *permissive planning*. Explicit approximations are employed in representing the world state and operators. Plans are then constructed efficiently using the approximate theory. In response to plan execution failures, plans are refined so they become less sensitive to the approximate knowledge used in their initial construction. This is achieved by tuning parameters of the plan so as to minimize the expected future deviation.

Each permissive plan has a target success rate and a degree of confidence desired in that success rate. We present a formal permissive planning algorithm which can be shown to either produce a plan with the desired success rate and degree of confidence, if possible, or otherwise to return the plan falling short of the target but with the best possible success rate. One of the downsides of this is that many examples are needed to gather the statistical evidence necessary to ensure these claims. Consequently, we propose an approximation to this algorithm which uses heuristics to determine how to refine plans and achieves good performance in the real-world domains we have investigated. We demonstrate the technique on the task of grasping of novel laminar objects and on orienting parts in a tiltable tray.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Gerald DeJong, for directing this research. I benefitted greatly from my many discussions with him as the work progressed. His unique perspective on learning and planning and enthusiasm for this project acted as a strong catalyst for this work.

I thank Professor Seth Hutchinson for his discussions with me on robot planning, his enlightening comments on how roboticists view the world, and for his suggestions on how to make my work more accessible to the robotics community.

I would also like to thank Professors Ken Forbus, Stephen Lu, and Thomas Huang for serving on my final committee and providing helpful comments.

Past and present members of the Explanation-based Learning group provided for a friendly, enjoyable atmosphere in which to conduct the work. I would like to thank Dr. Steve Chien, Melinda Gervasio. Jon Gratch, and Dan Oblinger as friends and for the useful feedback they gave me on this work. Many others in the Beckman Artificial Intelligence group were also helpful. I would especially like to thank Thomas Uribe and David Page for their useful critiques of the formal logic I employed at various stages of this work. I would also like to thank Michael Barbehenn, Dr. Tim Chou, Professor John Collins, Hyeon-Kyeong Kim, Dennis DeCoste, Steve Lavalle, Sandeep Pandya, Eduardo Perez, Ilan Shimshoni, Gordon Skorstad, and Larry Watanabe. I also thank Dr. Brian Falkenhainer, Dr. Brad Whitehall, Bharat Rao and Professors Larry Holder, Ray Mooney, Shankar Rajamoney, Paul O'Rorke, Alberto Segre, and Jude Shavlik for comments and/or discussions in the early stages of this work.

I am very grateful to my parents for the love, support, and encouragement they provided through the many years it took to complete this work.

Most of this research was conducted at the newly constructed Beckman Institute at the University of Illinois. The Beckman Institute provided ideal facilities for this work. We set up a robotics laboratory to test our approach and employed the Beckman machine shop for manufacturing camera mounts and the tray used in the experiments. I would especially like to thank Bruce Marshall for his dedication to the needs of the researchers at Beckman and for his help in making our robotic laboratory a success. Some of the equipment used in this work was purchased jointly by the Artificial

Intelligence and Cognitive Science groups at Beckman with the plan of setting up a laboratory where AI and COGSCI techniques could be tested through interaction with the real world. I hope mine is the first of many projects which can employ the laboratory to test methods for interacting with the real world. I am also grateful for the newly formed Digital Video Laboratory at Beckman, which I used in editing video-taped demonstrations and for capturing some of the photos for this thesis. I thank IBM for donating the IBM RT125s and RT135s used in preparing this thesis and developing the implementation. Thanks to INTERLEAF for making their great desktop publishing program, used for this thesis, available to universities free of charge.

This research was conducted at the Coordinated Science Laboratory and at the Beckman Institute at the University of Illinois at Urbana-Champaign. Funding was provided through the Office of Naval Research under grants N00014-86-0309 and N00014-91-J1563.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

1. INTRODUCTION

For many years, planning researchers have worked on developing domain-independent classical planners. In classical planning, a sequence of actions is found through use of a domain theory, which, when applied from an initial state, yields a desired goal state. Most classical planners work in highly simplified domains often called *micro-worlds*. Such micro-worlds facilitate investigation of important planning issues while shielding systems from real-world complexity and uncertainty. However, even in such limited micro-worlds, it has been difficult to efficiently apply general planning techniques. Furthermore, micro-world plans generally perform very poorly in the real world, because the highly simplified theory on which they are based does not take into consideration a range of important factors of the real world. With classical planners, a tradeoff exists between the tractability of generating plans and the performance of the generated plans in the real world.

It is often possible to use a micro-world theory to find several plans accomplishing the same goal from the same initial state. While there is no difference in what these plans accomplish with respect to the micro-world theory, there may be a profound difference in what they accomplish in the real world beyond the resolution of the micro-world theory. It is therefore reasonable to consider that some of these plans work better than others with respect to our goals in the real world. Given a failure of one plan, one should consider another of the alternative plans. This is an especially desirable option if information from the failure can help to guide us to a better alternative plan. Furthermore, in this way a plan has been "discovered" which yields good real-world performance without the need to resort to expensive sensing and reactivity. We call this new approach *permissive planning*. *Permissiveness* is a measure of how faithfully a plan's preconditions need to reflect the world in order to accomplish its goals. One way to increase plan permissiveness is to have the plan continue to adhere to the micro-world theory while tuning it to increasingly accomplish its goals in the real world.

As we discussed above, a given micro-world theory may support several different action sequences which accomplish the same goal from the same initial state. One of these action sequences, each of which constitutes a different plan for accomplishing the same goal, may include a variety of discrete and continuous ordered parameters. Permissive planning involves tuning those parameters in response to encountered failures. This is done in such a way that the resulting refined plan still accomplishes the goal in terms of the micro-world theory. Thus permissive planning explores alternative settings for the parameters within a plan without altering the structure of the plan. Different action sequences which achieve the same goal are treated as separate plans and are tuned separately.

If the process is successful, the refined plan is uniformly more permissive than the original, which it replaces. Thus, through interactions with the world, the system's plan library becomes increasingly permissive, reflecting a tolerance of the particular discrepancies that the training problems illustrate. This, in turn, results in a more reliable projection process. Notice that there is no improvement of the projection process at the level of individual operators. Performance improvement comes at the level of plans whose parameters are adjusted to make them more tolerant of real-world uncertainties in conceptually similar future problems. Adjustment is neither purely analytical nor purely empirical. Improvement is achieved through an interaction between background knowledge and empirical evidence derived from the particular real-world problems encountered.

The notion of permissive planning is not tied to any particular domain. It is a domain-independent notion that is, nonetheless, not universally applicable. There are characteristics of domains, and problem distributions within domains, that indicate or counter-indicate the use of permissive planning. Later, the requirements of permissive planning will be made more formal. Here we give an intuitive account of characteristics needed to support permissive planning. An application that does not respect these characteristics is unlikely to benefit from permissive planning.

For permissive planning to help, internal representations must be reasonable approximations to the world. By this we mean that there must be some metric for representational faithfulness, and that along this metric, large deviations of the world from the system's internal representations are less likely than small deviations.

Next, for a given goal and initial state, some planning choices for the values of discrete and continuous ordered quantities must not be fully constrained. These ordered quantities are called *parameters* of the plan. A set of constraints and preferences on parameter values must exist, relating them to the values of explicitly approximate quantities. These constraints and preferences are tuned, using information obtained through failures, in order to decrease the likelihood of similar failures.

Lastly, the domain must be one in which failures can be tolerated during the learning phase. A permissive planner improves over time, reaching some peak success rate for a given distribution. In domains where failures are prohibitively expensive, more expensive guaranteed planning approaches are warranted.[1]

One of the types of domains in which these requirements are met is in robotic manipulation. Consider a domain in which an overhead automatically controlled gantry arm is used to place and move boxes in a warehouse as shown in Figure 1.1. Boxes can be picked up and put down from above and are placed in stacks throughout the warehouse. Using a simple domain theory, a plan is constructed to move a box from the top of one stack to a destination stack. This consists of moving the manipulator from its original position to the top of the box to be moved, gripping the box, lifting the box to an appropriate height for the move, moving the box to a point above its destination stack, moving the box down to rest on the top of the stack, and ungripping the box. At the most abstract level, this is similar to a micro-world domain known as the *blocks world* where blocks are moved from one stack to another stack within a row of stacks. However, our operators for moving and grip-

1. Alternative approaches, particularly in the field of robotic planning, are discussed in Chapter 6.

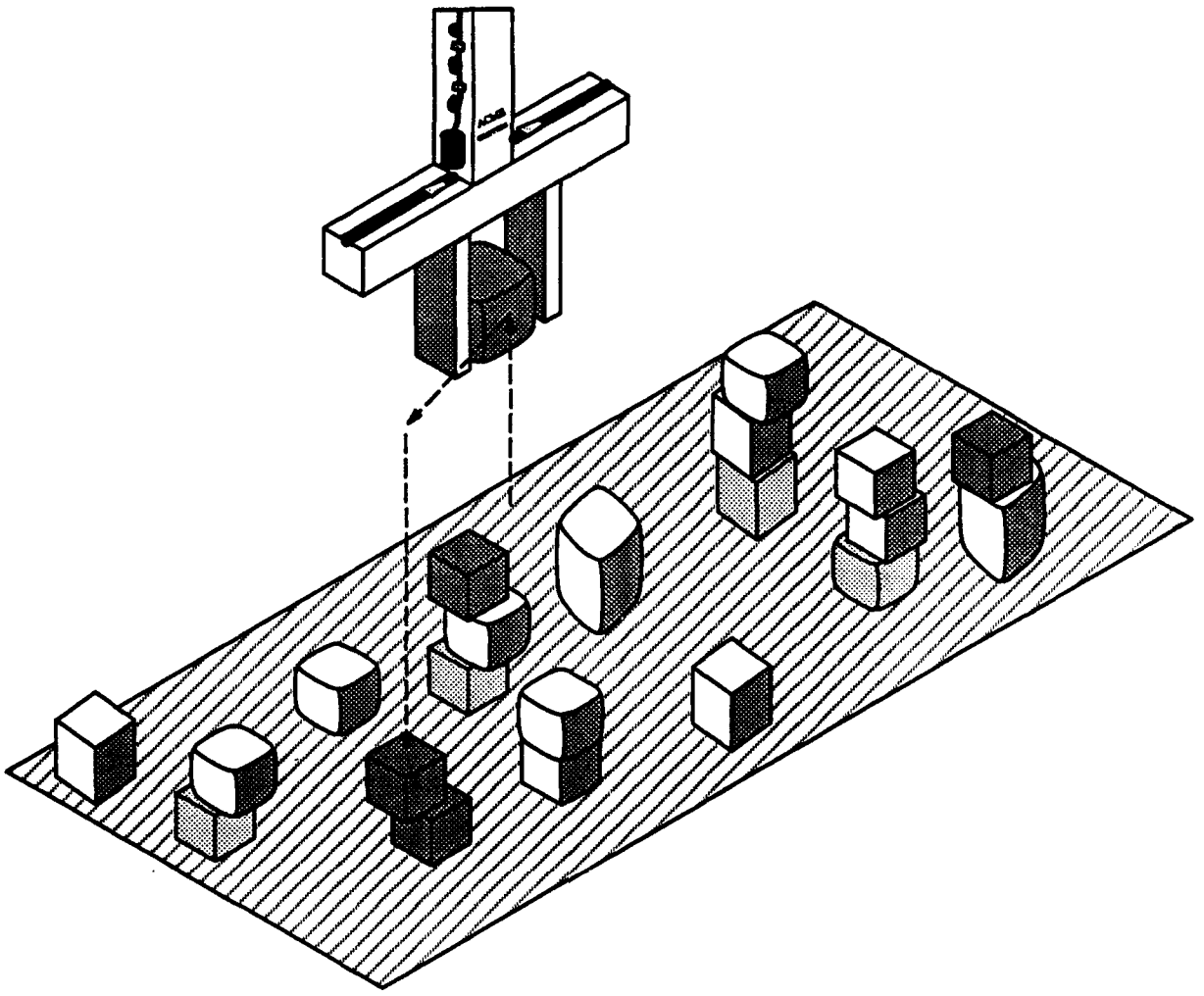


Figure 1.1. Automated Box-Moving on a Warehouse Floor

ping the boxes are realized with actual numeric positions, opening widths, forces, and so forth. After all, such values are necessary to command a real-world gantry arm. This also means that our micro-world model for actions in this domain often only partially constrains many of these values. In transporting a box from one location to another, the plan must guarantee, in terms of the micro-world model of the state of the warehouse, that no other pile will be knocked over in traversing the space. That is, the arm must lift the box being moved high enough not to collide with any of the other piles. It also has constraints on how high it may lift it due to constraints on the arm itself. Otherwise, heights between the bounds all satisfy the goal of allowing the box to be safely moved. Similarly, in gripping a box, a specific amount of pressure must be applied to ensure that the box does not slip away and fall to the floor. If too much pressure is applied, the box may be punctured. **Clear-**

ance-height and **gripping-force** are two of many parameters which play a part in our simple box moving plan.

It is also important to notice that good settings for these parameters are not easy to obtain without experience. The system must rely on sensors to indicate the state of the world. An incorrect reading of the height of a stack may result in a collision. An incorrect estimate of the strength of a box may result in its being crushed. Also, tradeoffs exist with regard to parameter settings. A plan which raises the box being moved to an extreme height to avoid hitting others wastes time that could be better spent moving more boxes. In fact, an excessively slow plan could be just as much of a failure to the warehouse manager as one which drops a box.

Ordinary classical planning techniques, in using a micro-world theory for plan construction, yield plans with poor real-world performance. Permissive planning critically relies on a learning phase where ordinary micro-world plans are tuned to yield improved performance. Therefore, in order to employ permissive planning, it must be possible to tolerate failures which occur during this learning phase. Since failures are used for learning, it must be possible to recognize failures and to obtain information about them. To make this possible, micro-world plans are augmented with a set of sensor expectations for every action in the plan. The sensor expectations for an action define a profile which the readings of specified sensors must follow during execution of the action. A failure occurs when the sensor readings do not follow the expected profile. Sensing is used during plan execution when the permissive planner is seeking to improve plan performance. The sensory information is being used to detect failures and to serve as a pointer to elements of the plan which need improvement.

In our gantry arm example, a sensor which can detect a collision between the box being carried and nearby objects would be ideal for detecting failures in traversing the warehouse with a box. One action of the box-moving plan might be a straight-line move of the arm while carrying the box high

above the floor. A simple expectation for this moving action would be that no collision be registered during the move.

When a failure occurs, the permissive planner produces a set of failure hypotheses. For the permissive planner, each failure hypothesis is rooted in some *explicit approximation*. An explicit approximation is an argument to some atomic formula in the domain theory whose potential values come from an ordered set of possibilities. That argument is explicitly marked as containing an approximation to some true, unmeasurable value. For instance, the coordinate values for the location of boxes in our warehouse example are explicitly approximate. Since our sensors are imperfect, they return explicitly approximate values. In real-world domains, large numbers of explicit approximations conspire to make the construction of working plans a difficult task. In our example domain, information about the arm location, location of other boxes, size of other boxes, force applied by the arm, and many other factors are all approximate. Every action in a permissive plan is justified in a support structure for that plan. Each failure hypothesis indicates one way in which the supports for the failing action might not hold due to a bad explicit approximation. For instance, the box collision may have resulted because the position of the box on the top of a nearby stack was estimated incorrectly.

Given a failure hypothesis, the permissive planner generates a tuning hypothesis which describes how some ordered parameter of the plan may be tuned to decrease the chance of the encountered failure. In the box collision case, this may be as simple as keeping the arm farther from nearby stacks while moving. In the case of a box being crushed while squeezing it, the applied force is decreased. Naturally, tradeoffs occur. If too little force is applied, the box may be dropped. More complex tradeoffs occur in other domains: for instance, in planning a grasp for an object with a complex shape, a task we will later investigate.

The permissive planner implements a tuning hypothesis with respect to the failing plan. Should the tuned plan still fail, more failure and tuning hypotheses are generated. It is possible that the permis-

sive planner cannot generate a working plan because of the domain theory it was given. However, we will show through demonstration in two significant real-world domains, robotic grasping and part orientation, that with a simple, easily generated theory, permissive planning significantly improves performance of micro-world plans.

In Chapter 2, we present a theory and algorithm for permissive planning. Here we will give precise definitions for plan parameters and explicit approximations and define the requirements for failure and tuning hypotheses. The algorithm we present produces a plan with the desired success rate and confidence (if attainable) by performing a statistical comparison of alternative plan tunings. In Chapter 3, we give a tractable approximate implementation to the theory and algorithm of Chapter 2. The approximation gains speed and expressive power but lacks the rigorous statistical validation of the earlier algorithm. Chapters 4 and 5 present implementations and empirical results in a robotic grasping domain and part orientation domain, respectively. Related work is discussed in Chapter 6. Conclusions and future directions for this work are presented in Chapter 7.

2. A THEORY OF PERMISSIVE PLANNING

Classical planning techniques make use of a micro-world theory to formulate a set of actions to achieve a goal. However, the micro-world domain theory, as the term micro-world indicates, is but an approximate description of the way the world really behaves. For planning to be tractable, one can never escape dealing with approximate theories. The more faithfully one tries to have the domain theory reflect the true behavior of the world, the less tractable planning becomes. Plans formulated using a micro-world domain theory often fail when carried out in the real world. These failures occur because of discrepancies between the micro-world theory and the behavior the real world exhibits. Unfortunately, these problems have lead many researchers to abandon this classical approach altogether. Instead, techniques which use little or no classical projection have been advocated such as reactive planning, probabilistic models, and neural networks. What is really needed by planning systems are simple yet effective domain theories for the construction of plans with good real-world performance. In this chapter, we give a formal presentation of *permissive planning* which addresses this need. We will define what it means to be a *permissive planner* and will describe the conditions which must be met for permissive planning to be used.

2.1. A Model of Real-World Planning

To make it easier to describe permissive planning, we will first define a model of real-world planning. Let a *generalized plan* consist of a goal specification and an operator sequence designed to achieve the goal. The goal specification is a partial state description and its inclusion with our generalized plan represents a departure from traditional plan definitions. Operators in the general plan's operator sequence may include variables in their specification. For the operators to be carried out in the world, these variables must all be assigned values. We will use the notation $GP|_{\vec{V}}$ where $\vec{V}$ is a vector of values $V_1 ... V_m$ to indicate a general plan GP whose operator sequence has been instan-

tiated under substitution $\{V_1/PV_1, V_2/PV_2, \ldots, V_m/PV_m\}$[2] where $PV_1 \ldots PV_m$ are the m variables of the plan's operator sequence.[3] Let $project(GP|_{\vec{V}}, s)$ be the projected state which results from applying the operator sequence of plan $GP|_{\vec{V}}$ to state s. Let $A(s)$ denote the (unknowable) actual state set for any state s sensed by the system.[4] Let $actual(GP|_{\vec{V}}, S)$ be the actual set of states which results from applying plan $GP|_{\vec{V}}$ to the actual states in state set S. The notation $Goal(GP)$ will be used to refer to the set of states given in the goal specification associated with plan GP. Now we can define three important sets: the first gives the states and variable values where general plan GP is projected to achieve the goal; the second gives the states and variable values where general plan GP actually achieves the goal in the real world, and the third gives the states and variable values where general plan GP is both projected to and actually achieves the goal.[5]

Definition 2.1: $M_{GP,Projected}$ is the set:

$$\left\{ < s_i, \vec{V}, s_f > \mid \left(s_f = project(GP|_{\vec{V}}, s_i)\right) \wedge \left(s_f \in Goal(GP)\right) \right\}$$

Definition 2.2: $M_{GP,Actual}$ is the set:

$$\left\{ < s_i, \vec{V}, s_f > \mid \left(s_f = project(GP|_{\vec{V}}, s_i)\right) \wedge \left(actual(GP|_{\vec{V}}, A(s_i)) \subseteq Goal(GP)\right) \right\}$$

---

2. The notation used is that of Nilsson [Nilsson80 , p. 141].

3. More generally, we can take vector V to be a set of planning commitments made by the planner in formulating a fully specified action sequence. Such commitments made by planners normally include variable assignments, codesignation constraints, and ordering constraints. We will be addressing fixed sequences and thus will not be discussing ordering constraints. We also will not address disjunctive sets of codesignations (these could be represented by separate generalized plans). Any nondisjunctive set of codesignations can be realized by renaming variables in the generalized plan. This work therefore focuses on the remaining planning commitment: variable assignment.

4. This is a set because there may be more than one real-world state which is labelled as the same state by a system's sensors.

5. The final state s_f in each of the tuples in these definitions is always the projected final state not the actual final state. In these definitions, it is also assumed that any action sequence can be attempted from any initial state.

Definition 2.3: $M_{GP,Ideal}$ is the set:

$$\left\{ x \mid x \in \left(M_{GP,Projected} \cap M_{GP,Actual} \right) \right\}$$

$M_{GP,Ideal}$ will always be a (possibly improper) subset of $M_{GP,Projected}$ because $M_{GP,Ideal}$ adds the constraint that the actual resulting state satisfy the goal. Ideally, a planning system will only choose to apply a plan GP with the initial state and variable assignments from an element of $M_{GP,Ideal}$. In this way, real-world achievement of the goal always results when the plan is applied. However, the actual mapping $actual(GP|_{\vec{v}}, A(s_i))$ (from Definition 2.2) and thus the set $M_{GP,Ideal}$ are not known to the planner. The set $M_{GP,Ideal}$ must be learned from feedback the planner obtains in applying plans in the real world. When the planning system starts out, it has no real-world experience. It must rely totally on plans derived from its theory of the domain. For a plan GP, the best information the planner has is that if the initial state and variable values are an element of $M_{GP,Projected}$, the plan will accomplish the goal. Of course, in real-world execution of the plan, errors in sensing the initial state, accurately controlling the actions, and shortcomings in the theory itself may cause the plan to fail. Through the experience of each failure, it is useful to identify some subset of the elements of $M_{GP,Projected}$ which, in fact, lead to real-world failures. In future applications of the plan, the planner should avoid choosing such variable valuations given by these failing elements.

2.2. The Plan Variable Space

Let S_{GP} be a space where each point corresponds to a fully determined version of plan GP. That is, the initial state to which the plan is applied is determined as is the action sequence and the final state predicted to result. The *projected region* of S_{GP} is the set of points in S_{GP} which correspond with fully determined plans which are *projected* to achieve the goal. That is, there is a one-to-one correspondence between points in the projected region and elements of the set $M_{GP,Projected}$. One

set of points in the projected region corresponds with the set of fully determined plans which will never succeed in the real world. Another set of points in that region corresponds with the set of fully determined plans which will always succeed in the real world (i.e., those which have a one-to-one correspondence with elements of the set $M_{GP\,Ideal}$).

A permissive planner seeks to find a description of the points at which the plan will succeed. The planner has a *target probability of success and coverage* (*TPSC*) as well as a desired confidence in that figure. By *probability of success* we mean the probability that the plan will succeed once it has been selected for application. By *coverage* we mean the probability that the plan will be selected for application from an initial state described in the projected region. Both factors are important for permissive planning. It is not very useful to have a plan which always works but seldom applies nor is it useful to have a plan which always applies but seldom works. For this reason, permissive planning seeks a combination of probability of success and coverage.

In this chapter, we will first define the vocabulary for descriptions of point sets in S_{GP}. Then, we will define a set of transformations which may be used to refine the descriptions. We will then describe a permissive planning algorithm which will either achieve the TPSC or will give the best probability of success and coverage short of the TPSC. This is accomplished through a statistically rigorous parallel evaluation of competing transformations.

2.3. The Vocabulary of Descriptions in S_{GP}

For each plan, a permissive planner seeks to achieve the plan's goal with the TPSC. It accomplishes this by identifying, if possible, some set of points in the projected region of S_{GP} where, by applying the plan, it will achieve this objective. We call such a set of points a *target region*. The projected region and target regions are described by a set of $2n$ constraints in S_{GP} where n is the number of dimensions in S_{GP}. Each constraint takes the form of an axis-aligned hyperplanar inequality (e.g.,

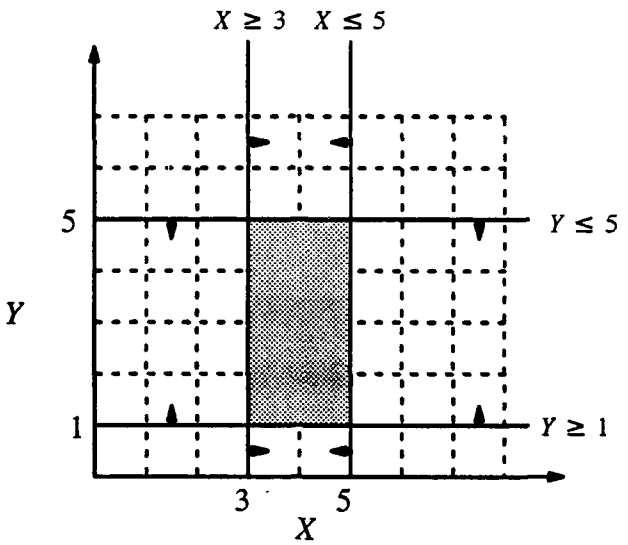


Figure 2.1. Four Conjunctive Constraints in a 2D S_{GP}

it describes a half-space of S_{GP}). Any set of constraints is taken as the conjunction of their respective axis-aligned hyperplanar inequalities. Figure 2.1 illustrates an example of four constraints in a two-dimensional S_{GP} with dimensions X and Y. The four constraints describe points inside a rectangle whose sides are parallel to the X and Y axes. The four inequalities are X≥3, X≤5, Y≥1, and Y≤5.

2.4. Transformations for Description Refinement

Every failure or success the system observes with a plan in the real world yields yet more useful information in finding a target region of S_{GP}. However, in analyzing real-world data, it may be unclear which of several failures actually occurred. Furthermore, for any single failure, there may be several candidate transformations to the current estimate of the target region. Finding a permissive plan which achieves the TPSC amounts to evaluating a set of possible transformations to this estimated target region description every time new information is obtained.

Every transformation entertained by the system consists of the modification of one of the constraints defining the estimated target region. Before a plan has been observed in the real world, the system's best guess of a target region is that it is equivalent to the projected region. Every refinement of this

estimate of the target region will embody a reduction in the estimated region's size. However, we will guarantee that each refinement made is a step forward to achieving the TPSC. Given a set of successes and failures of a plan, each transformation in the set of possible transformations to the estimated target region must satisfy:

1. The transformation must potentially affect system behavior. That is, the modification to the constraint must exclude some points formerly included in the estimated target region.

2. The transformation must be consistent with a tuning hypothesis for at least one of the observed failures.[6]

This definition for available transformations is more specific than simply allowing modification of all possible axis-aligned hyperplanar inequalities in classifying the observed successes from the observed failures. However, it is specific in a way that allows benefit to be gained from tuning hypotheses obtained directly from the original theory. These tuning hypotheses will be defined next.

Failures of the plan to accomplish the goal in the real world indicate a discrepancy between the theory on which the plan is based and the way the real world behaves. We introduce *explicit approximations* to model these discrepancies. The purpose of an explicit approximation is to mark a particular value as inexact. The value must be one of an ordered set of values. Every explicit approximation also embodies a characterization of inexactness: that the error in the inexact value is more likely small than large. More precisely, if V_a represents the approximate value of variable V and V_t is its true value, it must hold that: $P(\ V_t = z \mid V_a = x\ ) < P(\ V_t = z \mid V_a = y\ )$ if and only if $\left| y - z \right| < \left| x - z \right|$. When a plan fails, the failure is assumed to originate with some single ex-

6. Although, we could be more conservative in forcing each transformation to be consistent with at least one tuning hypothesis for each observed failure, this does not give us the flexibility to disregard one or more of the failures. The statistical analysis of the transformations may well permit a transformation, which does not account for some of the failures and successes, to meet our criteria anyway.

plicit approximation. Permissive planning is only driven by failures attributable to these explicit approximations.

In a general plan, each action has associated sensory expectations which must be justified by a supporting explanation. A failure of a fully determined version of a general plan is defined as a failure for the sensory expectations to hold in the real world. Each tuning hypothesis for the failure:

1. is based on a hypothesized failing explicit approximation employed in the supporting explanation for the failing expectations and

2. gives a suggested direction of movement (parallel to an axis) in S_{GP}, from the point corresponding to the failed plan, which is hypothesized to reduce the chance of the observed failure.

Given such a tuning hypothesis, a consistent transformation must exclude some of the points in S_{GP} in the half-space defined by a plane perpendicular to the suggested direction of movement on the side away from the suggested direction of movement. Figure 2.2 gives an example of a transformation in a two-dimensional S_{GP} with axes X and Y. The illustrated transformation modifies the constraint that values of X be less than or equal to 5 to that they be less than or equal to 5–ε, because

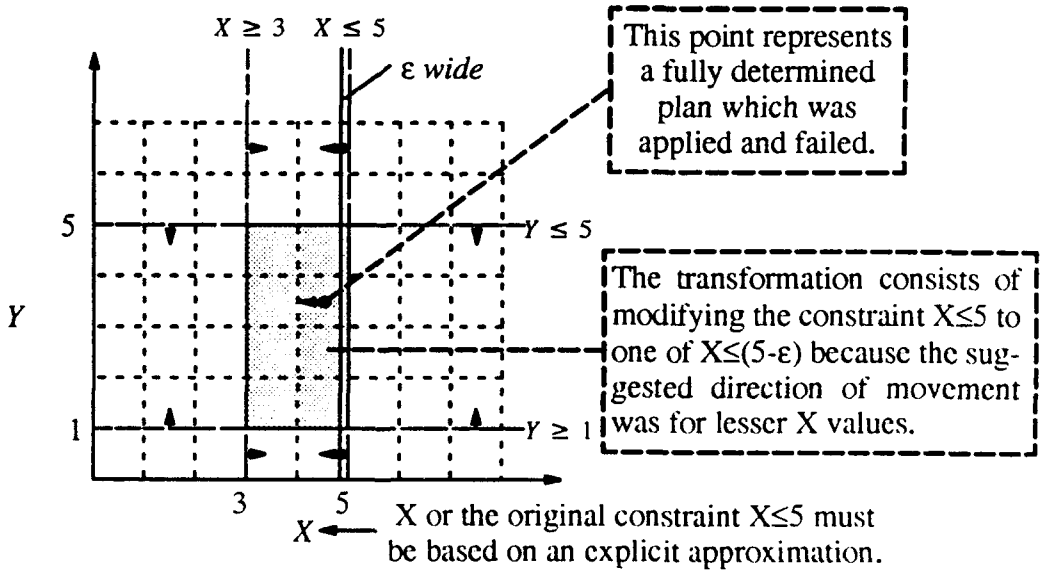


Figure 2.2. An Example Transformation in a 2D S_{GP}

the tuning hypothesis suggests that a smaller value of X is likely to reduce the chance of the observed failure. If a tuning hypothesis suggests this direction of movement then X itself and/or the value 5 in the constraint $X \leq 5$ is based on an explicit approximation. We will now give a more precise definition of the tuning hypothesis.

Let $EA(S_{GP}) = \{\ i \mid$ values along the ith dimension of S_{GP} are explicitly approximate$\}$. Let *ETR* (for *estimated target region*) be a set in S_{GP} such that $ETR \subseteq$ *Projected Region*. Let $L(i,ETR)$ and $U(i,ETR)$ be the lower and upper bounds, respectively, along the ith dimension for *ETR* (i.e., if $\vec{V} \in ETR$ then $L(i,ETR) < V_i < U(i,ETR)$ for all i). In addition to declaring all values along S_{GP} dimensions given in $EA(S_{GP})$ to be explicitly approximate, independently we allow that bounds on ETR along a particular dimension may be declared explicitly approximate. Let

$$EAL(ETR) = \{\ i \mid L(i, ETR) \text{ is explicitly approximate}\} \text{ and}$$

$$EAU(ETR) = \{\ i \mid U(i, ETR) \text{ is explicitly approximate}\}.$$

Let $TH(S_{GP}, ETR)$ (for tuning hypotheses associated with the fully instantiated plan corresponding to some $\vec{V} \in ETR$ in S_{GP}) be the set of 2-tuples:

$$\left\{ < i, D > \;\middle|\; \begin{array}{l} \Big((D = decrease) \wedge \big(i \in EA(S_{GP}) \ \vee \ i \in EAU(ETR)\big)\Big) \vee \\ \Big((D = increase) \wedge \big(i \in EA(S_{GP}) \ \vee \ i \in EAL(ETR)\big)\Big) \end{array} \right\}.$$

Each tuple gives the dimension of ETR to tune and the direction to tune it in D.

In Theorem 2.1 below we show that the set $TH(S_{GP}, ETR)$ is complete in if all failures are attributable to declared explicit approximations, the set contains all ways to potentially increase the probability of success through modifications to *ETR*.

Theorem 2.1: Completeness of $TH(S_{GP}, ETR)$

If the probability of success of a plan *GP* applied in region *ETR* can be increased by increasing *L(i,ETR)*, then $< i, increase > \in TH(S_{GP}, ETR)$, and if it can be increased by decreasing *U(i,ETR)*, then $< i, decrease > \in TH(S_{GP}, ETR)$.

Proof:

For every dimension i in S_{GP}, there are potentially four ways in which an explicit approximation can cause the plan to fail. Let $L=L(i,ETR)$, $U=U(i,ETR)$, and $V=V_i$ for a fully specified plan *GP* corresponding to $\vec{V} \in ETR$. Let a subscripted variable represent an explicit approximation with a subscript of 'A' indicating the approximate value and a subscript of 'T' indicating the true value. The four basic ways a single explicit approximation can cause a failure along dimension i are:

(1) $(V_A \leq U) \wedge (V_T > U)$ $\quad (< i, decrease > \in TH(S_{GP}, ETR))$

(2) $(V_A \geq L) \wedge (V_T < L)$ $\quad (< i, increase > \in TH(S_{GP}, ETR))$

(3) $(V \leq U_A) \wedge (V > U_T)$ $\quad (< i, decrease > \in TH(S_{GP}, ETR))$

(4) $(V \geq L_A) \wedge (V < L_T)$ $\quad (< i, increase > \in TH(S_{GP}, ETR))$

It is easy to verify for the four cases above that the parenthesized memberships to the right hold by our above definition $TH(S_{GP}, ETR)$.

Case 1:

Let $x < y \leq U < z$ where x, y, and z can be any values satisfying the relation. A failure of the plan corresponds to the case where $V_T=z$. Notice that it is always true that $|y-z| < |x-z|$. By definition of an explicit approximation, we can therefore conclude $P(V_T=z|V_A=x)<P(V_T=z|V_A=y)$. Lesser values for V_A are more likely to succeed and thus $< i, decrease > \in TH(S_{GP}, ETR)$.

Case 2:

Let $z < L \leq y < x$ and verify that $|y-z| < |x-z|$; therefore, $P(V_T=z|V_A=x)<P(V_T=z|V_A=y)$. Greater values for V_A are more likely to succeed and thus $< i, increase > \in TH(S_{GP}, ETR)$.

Case 3:

Let $x < y \leq V < z$ and verify that $|y-z| < |x-z|$; therefore, $P(U_T=z|U_A=x)<P(U_T=z|U_A=y)$. Lesser values for U_A are more likely to succeed and thus $< i, decrease > \in TH(S_{GP}, ETR)$.

Case 4:

Let $z < V \leq y < x$ and verify that $|y-z| < |x-z|$; therefore, $P(L_T=z|L_A=x)<P(L_T=z|L_A=y)$. Greater values for L_A are more likely to succeed and thus $< i, increase > \in TH(S_{GP}, ETR)$.

Given a failed fully instantiated plan GP corresponding to some $\vec{V} \in ETR$, $TransETR(S_{GP}, ETR)$ gives a new set of potential estimated target regions based on the possible tuning hypotheses. Let

n give the number of dimensions of points in S_{GP}. $TransETR(S_{GP}, ETR)$ is defined:

$$\left\{ ETR' \;\middle|\; EAL(ETR') = EAL(ETR) \wedge EAU(ETR') = EAU(ETR) \wedge \left[\begin{array}{l} \left[\begin{array}{l} (< i, decrease > \in TH(S_{GP}, ETR)) \wedge (L(j, ETR') = L(j, ETR) \text{ for } j = 1, n) \wedge \\ (U(j, ETR') = U(j, ETR) \text{ for } j = 1, n \text{ with } j \neq i) \wedge \left(U(i, ETR') = U(i, ETR') - \epsilon\right) \end{array}\right] \vee \\ \left[\begin{array}{l} (< i, increase > \in TH(S_{GP}, ETR)) \wedge (U(j, ETR') = U(j, ETR) \text{ for } j = 1, n) \wedge \\ (L(j, ETR') = L(j, ETR) \text{ for } j = 1, n \text{ with } j \neq i) \wedge \left(L(i, ETR') = L(i, ETR') + \epsilon\right) \end{array}\right] \end{array} \right] \right\}$$

Each transformation specializes the general plan by eliminating a class of fully specified plans using information from the failure of a single fully specified plan.

2.5. The Permissive Planning Algorithm

Each permissive plan has associated with it a target probability of success and coverage *TPSC* and confidence δ. We present an algorithm which will find either a region meeting the success criteria *TPSC* with confidence δ or one which falls short but finds the best value for probability of success and coverage. The algorithm is given below followed by an explanation and proof of correctness.

The permissive planning algorithm involves refining an estimated target region in a series of stages. The estimated target region (ETR) is a (possibly improper) subset of the projected region. Only fully specified plans GP corresponding to points in ETR will actually be applied. Let *TPSC* be the target probability of success and coverage for plan *GP* and let δ be the target confidence. The TPSC is a weighted combination of coverage and probability of success. *Coverage* is the likelihood of the planner applying the plan for a given point in the projected region. Probability of success is measured over cases in which the plan was actually applied. Let $TPSC = w_c C + w_s S_p$ where C is coverage, S_P is probability of success, and w_c and w_s are weights given on coverage and probability of success, respectively. Let τ be a small fixed threshold for the minimum improvement in probability of success between stages.

The following is the *PermPlan* algorithm which takes a generalized plan GP and an $ETR \subseteq ProjectedRegion \subseteq S_{GP}$ and returns a hyper-rectangular region $R \subseteq ETR$ satisfying the TPSC with the desired confidence δ or returns a region $R \subseteq ETR$ which gives the highest possible probability of success and coverage $R_{PSC} < TPSC$ with the desired confidence δ :[7]

```
R ← ETR                                          {save best ETR thus far}                    (1)
Util_n ← SignWithDelta1 (X_i − TPSC, δ, ETR)     {compare ETR with TPSC}                     (2)
if Util_n ≥ 0 then                                                                           (3)
   {ETR has achieved the desired TPSC and δ}                                                 (4)
else                                                                                         (5)
begin                                                                                        (6)
   ExploredETRs ← {}                             {no ETRs explored yet}                      (7)
   CandidateETRs ← { ETR }                       {initialize candidate ETRs for evaluation}  (8)
   done ← false                                                                              (9)
   repeat                                                                                    (10)
      CurrentETR←SelectOne(CandidateETRs) {select an ETR to evaluate}                        (11)
      NewETRs ← TransETR(S_GP, CurrentETR) − ExploredETRs                                    (12)
      GoodETRs ← PosWithDelta2 (X_ij − X_i − τ, δ, NewETRs, CurrentETR)                      (13)
      if ((GoodETRs = ∅) and Util_n ← SignWithDelta2 (X'_i − X_i, δ, CurrentETR, R)          (14)
         and (Util_n > 0) ) then R ← CurrentETR                                              (15)
      SufficientETRs ← PosWithDelta1 (X_ij − TPSC, δ, GoodETRs)                              (16)
      if (SufficientETRs ≠ ∅) then                                                           (17)
      begin                                                                                  (18)
         {R ← SelectOne(SufficientETRs) has achieved the desired TPSC and δ }                (19)
         done ← true                                                                         (20)
      end                                                                                    (21)
      ExploredETRs = ExploredETRs ⋃ {CurrentETR} ⋃ (NewETRs − GoodETRs)                      (22)
      CandidateETRs = (CandidateETRs ⋃ GoodETRs) − {CurrentETR}                              (23)
   until done or (CandidateETRs = ∅)                                                         (24)
end                                                                                          (25)
```

The first step in the algorithm (lines 1–4) is to apply the plan *GP* until sufficient evidence has been gathered to know with δ confidence whether the probability of success and coverage (*PSC*) of *ETR* exceeds the *TPSC*. If it does not, a set *CandidateETRs* (initially containing *ETR*) is formed containing candidate estimated target regions for refinement (line 8). While *CandidateETRs* is nonempty

7. The functions ***SignWithDelta1***, ***SignWithDelta2***, ***PosWithDelta1***, and ***PosWithDelta2*** are defined in the Appendix.

and no *ETR* has yet satisfied the *TPSC*, an element *CurrentETR* of *CandidateETRs* is selected for refinement (line 11). A set of refined versions of *CurrentETR*, called *NewETRs*, is formed by taking the set of possible refinements $TransETR(S_{GP}, CurrentETR)$ and removing any already explored *ETRs* (line 12). A set $GoodETRs \subseteq NewETRs$ is formed consisting of the *ETRs* with a probability of success and coverage (*PSC*) at least τ better than *CurrentETR* with δ confidence (line 13). If *GoodETRs* is empty, then a τ local maximum has been reached and *CurrentETR* is compared with the best known region *R* found thus far. If it is better than *R* with δ confidence, let *R* be *CurrentETR* (lines 14–15). If *GoodETRs* is nonempty, the set is checked for any element which exceeds the *TPSC* with δ confidence. If one is found, the algorithm returns it (lines 16–21). If none yet exceeds the *TPSC*, then members of *GoodETRs* are added to *CandidateETRs*, *CurrentETR* is removed, and we return to select another element to expand from the set *CandidateETRs* (lines 22–23). At algorithm termination, *R* will either be a hyper-rectangular region satisfying the *TPSC* with δ confidence or will be the hyper-rectangular region in the initial *ETR* with the highest *PSC* with δ confidence.

Let us calculate the number of *ETR* regions explored by the algorithm in the worst case. Let $ETR_{init} \subseteq ProjectedRegion \subseteq S_{GP}$ be the initial *ETR* given at the start of the algorithm. The algorithm refines the region in steps of ϵ. We can therefore calculate for a dimension i of ETR_{init} the number of positions available to place a bound:

$$n(i) = \left\lceil \frac{U(i, ETR_{init}) - L(i, ETR_{init})}{\epsilon} \right\rceil + 1 .$$

Let M_1 be the set of dimensions i in ETR_{init} along which only a single bound can be moved in a tuning hypothesis. This is given by

$$M_1 = \begin{matrix} (EAL(ETR_{init}) - EAU(ETR_{init}) - EA_{GP}) \cup \\ (EAU(ETR_{init}) - EAL(ETR_{init}) - EA_{GP}) \end{matrix} .$$

Let M_2 be the set of dimensions i in ETR_{init} along which both bounds can be moved by a tuning hypothesis. This is given by

$$M_2 = EA_{GP} \bigcup \left(EAL(ETR_{init}) \bigcap EAU(ETR_{init}) \right).$$

The number of possible regions which can be explored by the algorithm is then given by

$$MaxETR = \prod_{\forall i \in M_1} \left(n(i) - 1 \right) \prod_{\forall i \in M_2} \frac{n(i)\left(n(i) - 1\right)}{2}.$$

The first product term gives the number of ways to move a single bound for each of the dimensions in M_1, and the second product term gives the number of ways to move two bounds for each of the dimensions in M_2. The product of the two terms gives the maximum number of regions which can be explored by the algorithm.

Let us now consider the number of plan execution examples required by the algorithm in the worst case. Since all of the statistics gathered in the algorithm are on bounded values, let ST(δ) be an upper bound on the stopping time for the Nádas criterion employed in gathering examples for the ***SignWithDelta1***, ***SignWithDelta2***, ***PosWithDelta1***, and ***PosWithDelta2*** functions. That is, ST(δ) examples will be required in the worst case for each of these function calls. The total number of examples employed by the algorithm in the worst case is $ST(\delta) + 2\, n_e\, ST(\delta)$, where n_e is the number of regions expanded to possible refinements. If each region had only one refinement, n_e would be *MaxETR* resulting in a worst-case number of examples $ST(\delta) + 2\, MaxETR\, ST(\delta)$. Of course, the point in using the iterative stopping criterion in gathering statistics is to minimize the number of examples required to achieve the desired confidence goals. The number of examples required for a given δ is sensitive to the distribution.

Theorem 2.2: *Correctness of PermPlan Algorithm*

The PermPlan Algorithm takes a generalized plan GP and an $ETR \subseteq S_{GP}$ and returns a hyper-rectangular region $R \subseteq ETR$ satisfying the TPSC with the desired confidence δ or returns a region $R \subseteq ETR$ which gives the highest possible probability of success and coverage $R_{PSC} < TPSC$ with the desired confidence δ.

Proof:

If the initially given estimated target region *ETR* has a *PSC* exceeding *TPSC* with δ confidence, the algorithm terminates and returns *ETR* (lines 1 through 4). For every *CurrentETR* selection from *CandidateETRs NewETRs=TransETR(S_{GP},CurrentETR)* gives all possible new candidate estimated target regions. We showed in Theorem 2.1 that the set *TH(S_{GP},CurrentETR)* was guaranteed to include all possible ways of improving the probability of success of a plan GP corresponding to a point $\vec{V} \in CurrentETR$. *TransETR(S_{GP},CurrentETR)* gives a concrete way of implementing each tuning hypothesis by moving in the indicated bound on *CurrentETR* by a small amount ε. Each of the elements of *NewETRs* is evaluated (line 13) to guarantee that its *PSC* exceeds the *PSC* of *CurrentETR* by τ with δ confidence. The choice of values for ε and τ thus determines the grain size of the search for a target region. A smaller value for these gives a more accurate result at a higher computational expense. The algorithm will terminate because each candidate target region can be explored at most once and the number of candidate target regions is bounded above by *MaxETR* (shown earlier). If one of the *MaxETR* regions achieves the *TPSC* or gives the highest τ local maximum, it will eventually be recognized and returned by the algorithm.

It has been our goal in this chapter to precisely describe the permissive planning technique. It is also our goal to show the viability of the approach in some real-world domains. In the next chapter, we will discuss practical approximations to the algorithm outlined here which facilitate such a real-world implementation of the approach.

3. A PRACTICAL APPROXIMATION TO PERMISSIVE PLANNING

In this chapter, we describe a permissive planner called GRASPER,[8] which is an implemented approximation to the representations and algorithm as given in Chapter 2. It is an approximation in several ways. The space S_{GP} in our theory consisted of points each of which fully specified the initial state, settings of all plan variables, and projected resulting state. Regions, such as the projected region, in S_{GP} were bounded by axis-aligned linear constraints in the space. In our implementation we commit to a particular representation for permissive plans and define the dimensions of a reduced space S'_{GP} based on attributes of the particular plan. For more expressive power, general linear constraints are allowed in defining regions in S'_{GP}. The number of constraints and particular form they take now depend on the domain theory employed. Since constraints are not all axis-aligned, constraints on one parameter may affect the choice of another.

The same iterative statistical methods as presented in Chapter 2 are used to decide when plan refinement should occur. However, statistical sampling is not employed in determining which of several refinements to pursue. Such sampling, although valuable in making statistically sound decisions, can require a large number of examples. In the theory, all theoretically possible failures and refinements were explored and statistically analyzed. We first use the method of monitoring for expected outcomes during execution to narrow the search for a failed approximation. Each action in the plan has an expected outcome supported by a portion of the explanation supporting the overall plan. The hypotheses for failure begin with the portion of the explanation indicated by the action with an expectation failure. Beyond using expectations to focus, a key heuristic is employed which measures the likelihood of one approximation failure relative to another based on the distances their respective parameters would have to be in error to have failed. Lastly, rather than just learning the parts of the estimated target region to exclude as was done in the algorithm of Chapter 2, we learn the likely

8. The name GRASPER is due to the first domain on which the system was tested, that of robotic grasping. The robotic grasping domain is discussed in Chapter 4.

"best" values within the new estimated target region. This is done by using not only linear bounds to constrain the region but by imposing linear utility functions to guide selection of plan variable values for new plan applications. These combined methods provide an expressive but efficient permissive planning implementation.

We start by describing our particular representation for general plans and their supporting explanations. Next, we discuss the approximation to S_{GP}. The following sections explain how permissive plan application and refinement are implemented. Lastly, we present a method for maintaining a library of multiple permissive plans in various states of refinement.

3.1. General Plans and Explanations

Plan refinement is key to permissive planning. This refinement requires that plans be justified by a support structure called an *explanation*. The term explanation is used to refer to a logical proof that a particular example is an instance of some concept. Explanations have often been used in classification tasks [Mitchell86]. However, in the context of a plan, our meaning for an explanation is that it justifies how a sequence of actions achieves some goal. Our view of explanations for plans is consistent with that of Mooney where explanations are connected directed acyclic graphs of *units* [Mooney87 , Section 3.1]. Each unit is a connected directed acyclic graph in which the vertices are predicate calculus well-formed formulas. We distinguish two basic types of units: *inferences* and *operators*.

Inferences are Horn clauses consisting of a finite set of antecedent well-formed formulas (wffs) and a consequent wff. A directed path exists from each antecedent to the consequent in an inference unit. An example of an inference unit is shown in Figure 3.1. This unit gives the distance between two two-dimensional points.

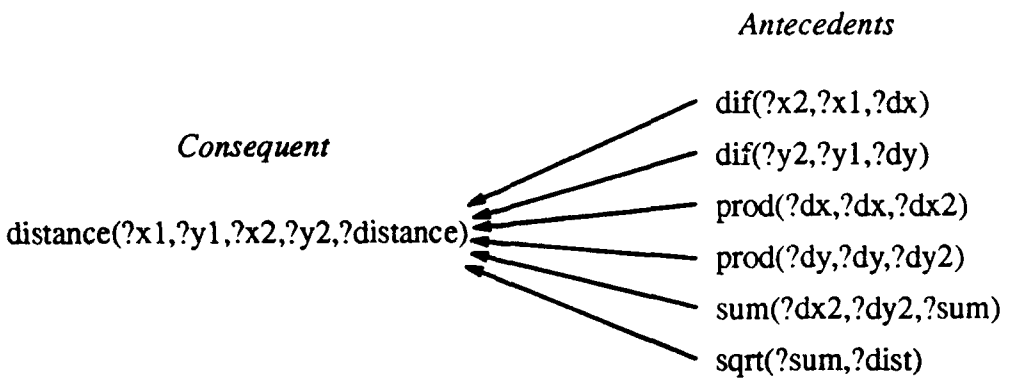


Figure 3.1. An Inference Unit

An operator unit has a finite set of precondition wffs, a finite set of effect wffs, and a ground atomic formula indicating the operator itself. Directed edges extend from all of the preconditions to the operator wff and from the operator formula to all effects. A simple operator for sending parcels from one location to another is shown in Figure 3.2.

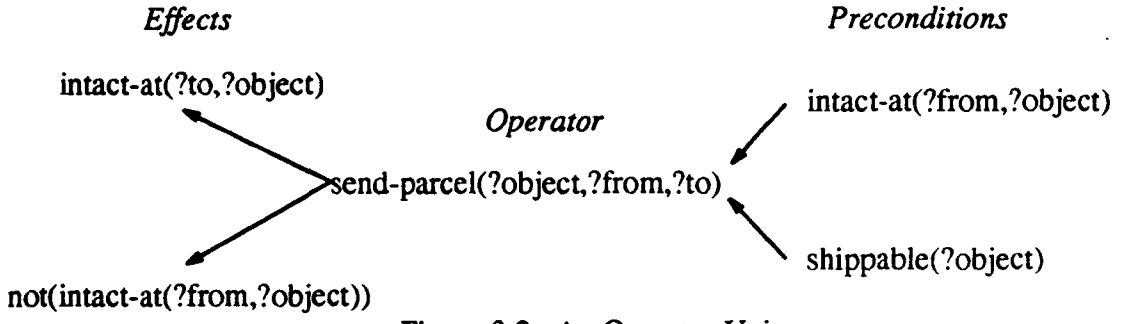


Figure 3.2. An Operator Unit

We refer to the wffs which are part of explanations as *facts*. Two important classes of facts are *state-defining* facts and *inferred* facts. State-defining facts are those asserted (or retracted) by operators. Operator effects must all be state-defining. Inference consequents are always inferred facts. The **intact-at** predicates of Figure 3.2 are examples of state-defining facts. By applying the **send-parcel** operator, a new state results where the object being sent is now in a new location and no longer at its former location. The consequent of the inference in Figure 3.1, **distance**, is an inferred fact.

Operators, such as the one illustrated, are used in conjunction with the STRIPS assumption [Fikes71]. The only effects the operator has on the world are those described by the operator effects. Negated effects such as **not(intact-at(?from,?object))** in the example are treated as members of the delete list are under STRIPS. Operators also have several other important fields not shown in the operator unit graph. These include termination conditions and expectations which should be met during operator execution. These are discussed in more detail when we discuss particular domains in Chapters 4 and 5.

Let us introduce an example of an explanation which will be used throughout the chapter to help in illustrating various requirements of the permissive plan. The example involves sending packages of various weights and sizes through the mail. A single operator **send-parcel(?object,?from,?to)** is employed for sending packages as was shown in Figure 3.2. The operator requires that the package start intact at the **?from** location and be shippable. As a result of operator application, the package is intact at the **?to** location and no longer at the **?from** location. In addition to the operator, we have several inference units shown in Figure 3.3.

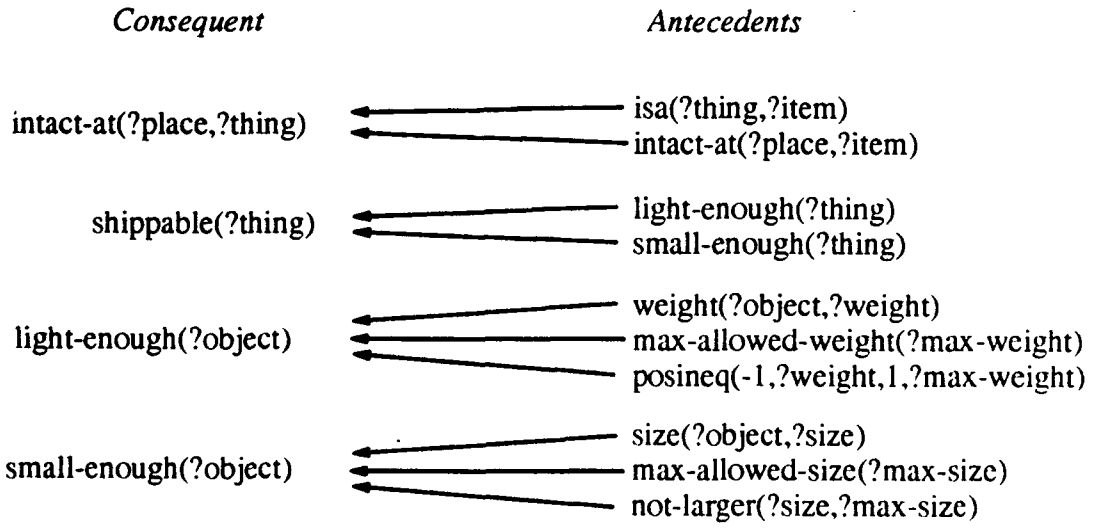


Figure 3.3. Inference Units for the Package-Sending Domain

The first inference expresses that if **?item** is a type of **?thing**, we have a **?thing** intact by having an **?item** intact. The second allows us to infer an object as shippable if it is light enough and small enough to be mailed. The third infers an object as light enough to be shipped if the weight is below the maximum weight which can be shipped. The last infers an object is small enough to be shipped if its size is less that the maximum allowable size. Sizes are members of the discrete ordered set {small, medium, large}. Weights are members of the set of positive real numbers.

We now add to our theory some facts about the domain. First, a discrete ordering on object sizes is expressed:

not-larger(small,small)
not-larger(small,medium)
not-larger(small,large)
not-larger(medium,medium)
not-larger(medium,large)
not-larger(large,large)

Next, three boxes are defined:

isa(container,box1)
intact-at(home1,box1)
weight(box1,10.2)
size(box1,small)

isa(container,box2)
intact-at(home1,box2)
weight(box2,15.1)
size(box2,large)

isa(container,box3)
intact-at(home1,box3)
weight(box3,23.7)
size(box3,medium)

Limits on the shipping weight and size are given as

max-allowed-weight(25.0)
max-allowed-size(medium)

Suppose that the desired goal is **intact-at(home2,container)**. There is more than one way in which the goal may be achieved and Figure 3.4 illustrates one possible explanation for goal achievement. The figure shows the *uninstantiated* explanation structure where arrows indicate supporting rela-

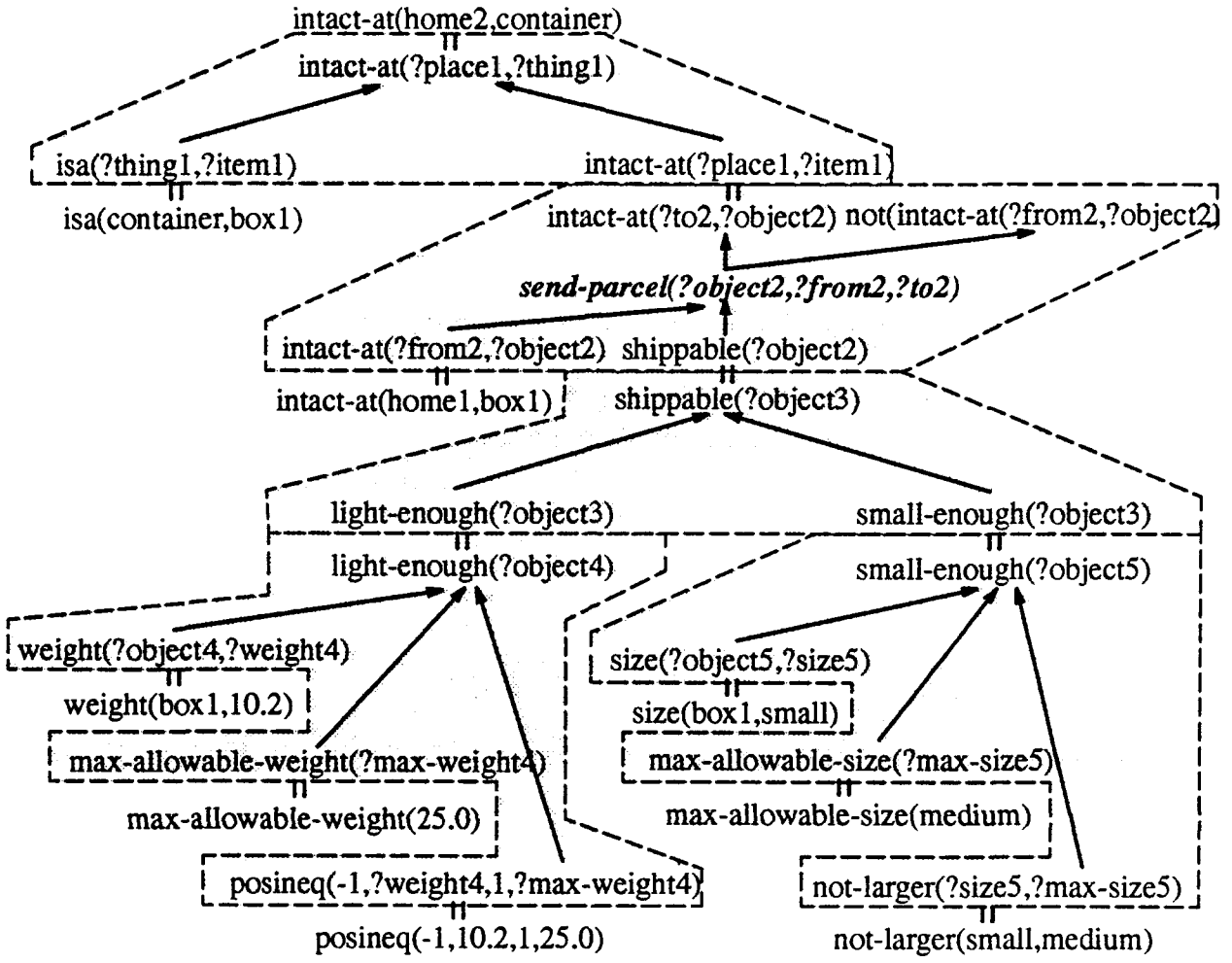


Figure 3.4. An Explanation for Achieving **Intact-At(Home2,Container)**

tionships between wffs and double bars indicate unifications. By *uninstantiated* we mean that units in the explanation structure are shown just as they were found in the domain theory but with uniquely named variables. An *instantiated* explanation structure would involve applying some substitutions of world objects for those variables. Each of the five shaded regions shows a unit (four inference units and the send-parcel operator unit). The wffs outside the units include the goal and all of the supporting facts required from the initial state. As a result of the unifications required to construct the proof for the specific case of sending box1, the variables have the following values:

?place1=home2,?thing1=container,?item1=box1
?object2=box1,?from2=home1,?to2=home2
?object3=box1
?object4=box1,?weight4=10.2,?max-weight4=25.0
?object5=box1,?size5=small,?max-size5=medium

The explanation of Figure 3.4 is for a very specific way for achieving the goal. We prefer to work with general plans and explanations which support those general plans and are not tied to specifics of an individual example. We therefore use the EGGS generalization algorithm to produce the general version of the explanation shown in Figure 3.5 [Mooney86]. This generalization preserves the structure of the explanation while abstracting away details of the example. The generalized plan associated with the explanation is shown in Figure 3.6. It is a macro-operator, just like the operators

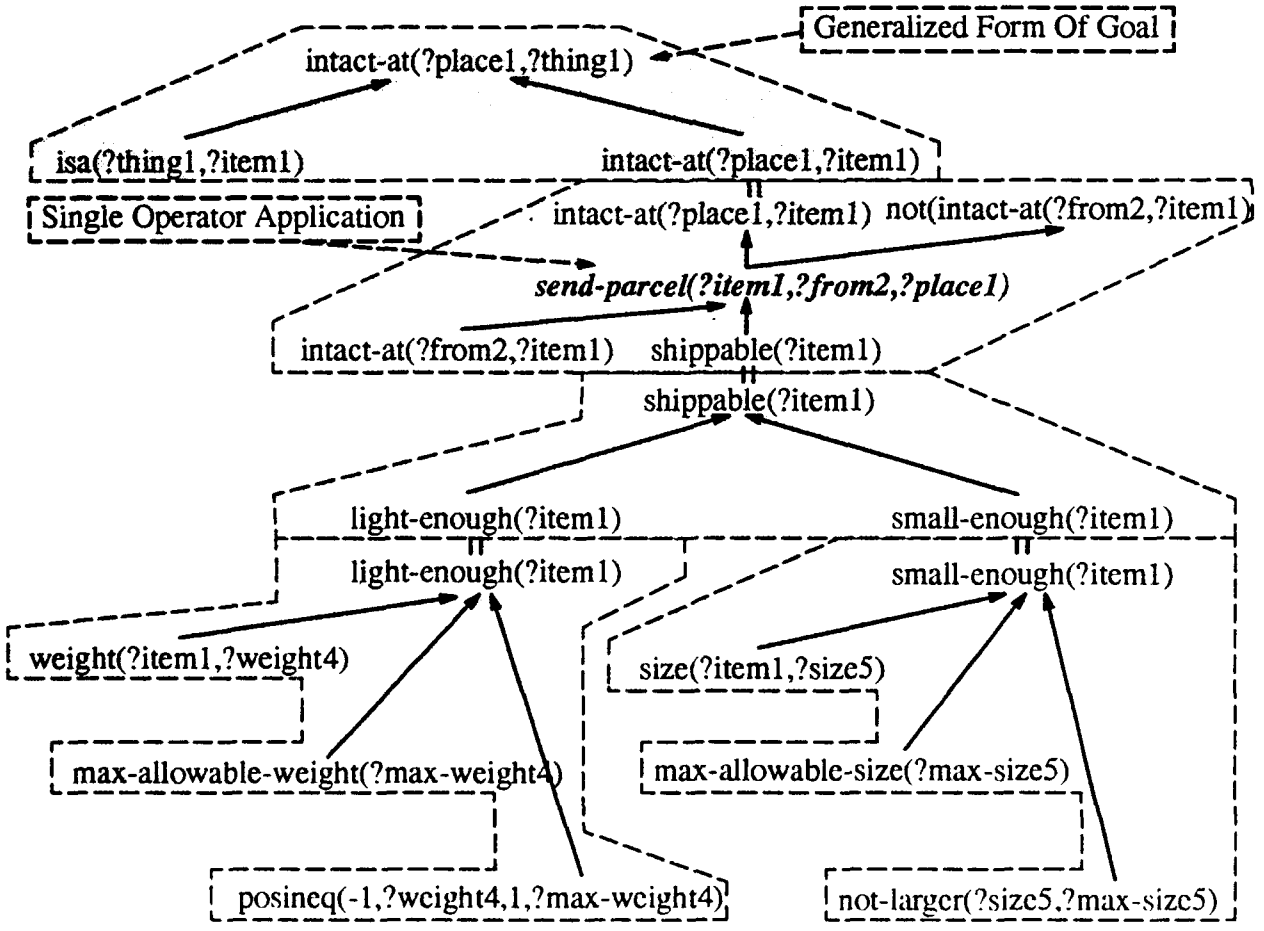


Figure 3.5. The Generalized Explanation Structure

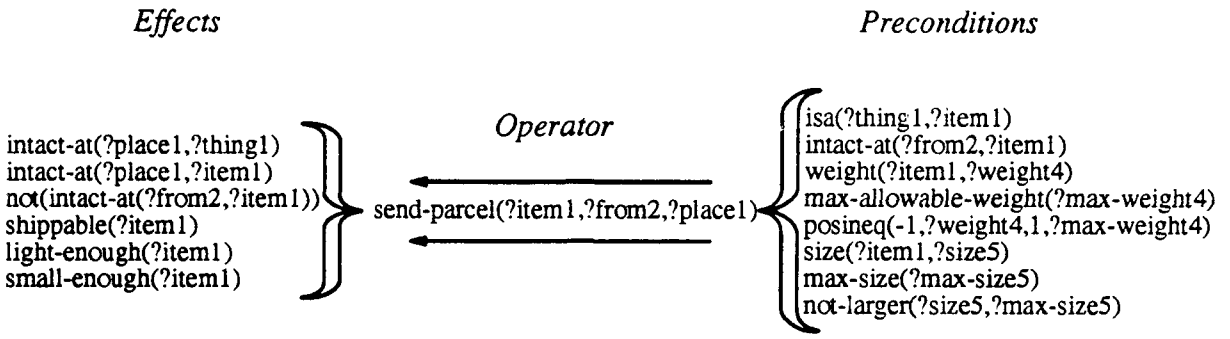


Figure 3.6. The Generalized Plan for Send-Parcel(?item1,?from2,?place1)

defined earlier except that it may include a sequence of actions rather than a single action. In this case, our plan performs only a single action: **send-parcel.**

As required by our theory of permissive planning given in Chapter 2, our implementation of a general plan includes both a goal specification (a partial state description) and an operator sequence to achieve that goal. Next, we introduce an approximation to S_{GP} and show how the projected region for the general plan can be described.

3.2. An Approximation to S_{GP}

Recall that in Chapter 2 the space S_{GP} is defined such that each point represents a fully specified version of the general plan with initial state, final state, and all plan variable values defined. We use the supporting explanation for the general plan (as described above) to find a set of plan parameters each of which constitutes a dimension of space S'_{GP} which approximates S_{GP}. First, we need to define some predicates and functions which work with the explanation to facilitate our definition for a parameter.

The predicate *Support(a,b,e)* holds if and only if a directed path exists from node *a* to node *b* in the explanation structure *e* (recall that an explanation structure is a directed-acyclic graph). Certain supporting nodes appear as leaves of the DAG and consequently have no node supporting them. These

are called *leaf supports*. *Leaf-Support(a,b,e)* holds if and only if ($Support(a, b, e) \wedge \neg \exists x \; Support(x, a, e)$). The set of leaf supports for a given node in the explanation is given by *Multiple-Leaf-Supports(A,b,e)* which holds if and only if $\forall_{a \in A} \; Leaf\text{–}Support(a, b, e)$.

Let *Predicate(f)* be a function which returns the predicate name of a fact *f* and let *Arguments(f)* be a function which returns the arguments to a fact *f*. Six predicates, $=$, $\leq$, $\geq$, *lineq*, *posineq*, and *negineq* are special in permissive planning and are called *constraints*. *Constraint(p)*[9] holds if and only if *p* is a constraint. Variables which appear as arguments to a constraint must all either take values from continuous ordered sets or from discrete ordered sets. *Ordered-Continuous(v)* holds if and only if *v* is a member of a continuous ordered set. *Ordered-Discrete(v)* holds if and only if *v* is a member of a discrete ordered set. Discrete constraints have 2 discrete arguments as with $\geq(large, small)$. Continuous constraints may be of linear form as with $posineq(2, a, 3, b, 4, c)$ to represent $0 \leq 2a + 3b + 4c$. We make a distinction between discrete and continuous constraints to facilitate our implemented approximation to the permissive planning algorithm.

It is useful to separate those leaf supports to a node in the explanation structure which are constraints with continuous arguments from those which are not. The former group of leaf supports is called *continuous leaf supports* and the latter is called *discrete leaf supports*. These are more formally expressed by the following predicates:

9. Second-order predicates such as *Constraint* are used only to simplify basic definitions and won't be employed by the system itself.

Continuous-Leaf-Support(c,f,e) ≡

c is a continuous leaf support of f in explanation structure e

for all c, f, and e:

Continuous-Leaf-Support(c,f,e) holds iff

there exists an S such that

$$Multiple\text{–}Leaf\text{–}Supports(S,f,e) \wedge c \in S \wedge Constraint(c) \wedge \forall_{a \in Arguments(c)} Ordered\text{–}Continous(a)$$

Multiple-Continuous-Leaf-Supports(C,f,e) ≡

C is the set of of continuous leaf supports of f in explanation structure e

for all C, f, and e:

Multiple-Continuous-Leaf-Supports(C,f,e) holds iff

$$\forall_{c \in C} Continous\text{–}Leaf\text{–}Support(c,f,e).$$

Discrete-Leaf-Support(d,f,e) ≡

d is a discrete leaf support of f in explanation structure e

for all d, f, and e:

Discrete-Leaf-Support(d,f,e) holds iff

there exists an S such that

$$Multiple\text{–}Leaf\text{–}Supports(S,f,e) \wedge d \in S \wedge (\neg Constraint(d) \vee \forall_{a \in Arguments(d)} Ordered\text{–}Discrete(a))$$

Multiple-Discrete-Leaf-Supports(D,f,e) ≡

D is the set of discrete leaf supports of f in explanation structure e

for all D, f, and e:

Multiple-Discrete-Leaf-Supports(D,f,e) holds iff

$$\forall_{d \in D} Discrete\text{–}Leaf\text{–}Support(d,f,e).$$

Let the function *Binding-Sets(F,s)* return the set of all variable substitutions $\{\theta_1, \theta_2, \ldots, \theta_n\}$ for which all facts $f \in F$ hold in state s. Let each of these substitutions be a most general unifier. Let

Possible-Bindings(v,B) be the set of possible substitutions for *v* across the substitutions in the substitution set *B*. For example, suppose the following facts hold:

contains(box1,object1)
contains(box2,object2)

Binding-Sets(contains(?box,?thing)) returns the set of possible substitutions:

{{box1/?box,object1/?thing},{box2/?box,object2/?thing}}.

The function *Possible-Bindings(?thing,Binding-Sets(contains(?box,?thing)))* returns the possible substitutions: {object1 object2}. The predicate *Unbound(v,B)* holds if and only if there is no substitution given for *v* in any substitution in the set *B*.

Let G be the goal of the plan (the sink of the explanation structure DAG) and let E_g be the generalized explanation structure. A parameter *p* for a state *s* is then defined as follows:

for all p, G, E_g, and s:

Parameter(p,G,E_g,s) holds iff
there exists a D and C such that:

$$\begin{aligned}
&\textit{Multiple–Discrete–Leaf–Supports}(D, G, E_g) \wedge \\
&(\; (\; \| \textit{Possible–Bindings}(p, \textit{Binding–Sets}(D, s)) \; \| \; > 1 \;) \vee \\
&\quad (\; \textit{Multiple–Continuous–Leaf–Supports}(C, G, E_g) \wedge \\
&\qquad \underset{c \in C,\ p \in \textit{Arguments}(c)}{\exists} \textit{Unbound}(p, \textit{Binding–Sets}(D, s)) \;) \;) \wedge \\
&(\; \textit{Ordered–Discrete}(p) \vee \textit{Ordered–Continuous}(p) \;).
\end{aligned}$$

A parameter is a variable whose values are taken from an ordered set and which can take on at least two different values while still supporting the goal in state *s*. It can either take on multiple values because more than one value is possible across the binding sets which support the plan explanation or because it has an under-constrained continuous value.

The package sending example introduced early in the chapter involves an explanation which includes discrete parameters. Recall that the goal was **intact-at(home2,container)**, that home2 has a container intact. The explanation of Figure 3.4 on page 27 provides one possible choice of boxes such that the goal is satisfied. In fact, there are two box choices which satisfy the goal. Consider our definition for plan parameter given above. In this case, the discrete leaf supports, D, for the goal, obtained from the generalized explanation structure shown in Figure 3.5 on page 28 are

isa(?thing1,?item1)
intact-at(?from2,?item1)
weight(?item1,?weight4)
max-allowable-weight(?max-weight4)
posineq(-1,?weight4,1,?max-weight4)
size(?item1,?size5)
max-size(?max-size5)
not-larger(?size5,?max-size5)

The set *Binding-Sets(D,s)* where *s* is the current state includes two substitutions and is

{{container/?thing1, box1/?item1, 10.2/?weight4, 25.0/?max-weight, small/?size5,
medium/?max-size5},
{container/?thing1, box3/?item1, 23.7/?weight4, 25.0/?max-weight, medium/?size5,
medium/?max-size5}}

Three variables within these binding sets have more than one possible binding: ?item1, ?weight4, and ?size5. While ?item1 represents the name of the box, its values are not drawn from an ordered set and thus cannot be considered a parameter. Both ?weight4 and ?size5 qualify as parameters as they have respectively continuous ordered and discrete ordered values. The identification of these parameters indicates that both weight and size of the box chosen may be tuned in response to failures of the box sending plan while still supporting achievement of the goal.

No underconstrained continuous parameters occur in our example. The only continuous leaf support for the goal in the generalized explanation is *posineq(-1,?weight4,1,?max-weight4)* and under both of the above binding sets, the arguments to this constraint predicate are fully specified.

A parameter is defined with regard to a specific state. Let *GlobalParameter(p,GP,E_g)* hold for parameter *p*, general plan *GP*, and explanation structure E_g if and only if there exists some state *s* such that *Parameter(p,GP,E_g,s)* holds. A parameter *p* corresponds to a dimension of S'_{GP} if and only if *GlobalParameter(p,GP,E_g)* holds. Every $\vec{V} \in S'_{GP}$ gives a value for every global parameter of plan GP. Each parameter V_i participates in one or more constraints given in the preconditions to the general plan. The conjunction of those constraints defines the *Projected Region* in S'_{GP}. Notice, that unlike the hyper-rectangles described by the constraints of the theory in Chapter 2, we only require that our linear constraints specify a single, nondisjunctive region in S'_{GP}. In the next section, we show how a $\vec{V} \in$ *ProjectedRegion* is selected for plan application.

3.3. Applying the Plan

We call the set of constraints which define the *Projected Region* in S'_{GP} *plan constraints* as they originate from the plan's preconditions. Additionally, the system imposes *learned constraints* learned through failures. For a continuous parameter, the following are possible forms for plan constraints:

1. A set of constraints of the form $lineq(a_1, x_1, a_2, x_2, \ldots, a_n, x_n)$ to represent $0 = a_1x_1 + a_2x_2 + \ldots + a_nx_n$ with n less than or equal to the number of parameters.

2. A set of constraints of the form $posineq(a_1, x_1, a_2, x_2, \ldots, a_n, x_n)$ to represent $0 \leq a_1x_1 + a_2x_2 + \ldots + a_nx_n$ with n less than or equal to the number of parameters.

3. A set of constraints of the form $negineq(a_1, x_1, a_2, x_2, \ldots, a_n, x_n)$ to represent $0 \geq a_1x_1 + a_2x_2 + \ldots + a_nx_n$ with n less than or equal to the number of parameters.

We also have at most four learned constraints for a parameter. Two of these are interval bounds on the parameter and one or two may serve to indicate the desirability of values within those bounds

by specifying a unimodal monotonic piecewise linear utility function of the parameter value. All the learned constraints are inequalities of the form specified in 2 and 3 above. We will return to how these constraints are learned and modified when we discuss plan refinement.

For discrete parameters, a similar set of simple constraints exists of the form: $= (x, y)$, $\leq (x, y)$, and $\geq (x, y)$. As many as four learned constraints exist for discrete parameters as well. Since discrete parameter values are members of ordered sets, and the arguments to a constraint must be members of the same ordered set, their set ordinals are used for $=$, $\leq$, and $\geq$ comparisons.

Let $U = w_1u_1 + w_2u_2 + \ldots + w_nu_n$ be an overall linear utility function combining the individual utilities $u_1, u_2, \ldots, u_n$ of the plan parameters with respective weights $w_1, w_2, \ldots, w_n$. A set of parameter values is sought which satisfies all of the plan and learned constraints while maximizing the overall linear utility function U. The algorithm is as follows:

Let *Multiple–Discrete–Leaf–Supports*(D, G, E_g) hold
Let P be the set of all p satisfying *Parameter*(p, G, E_g, s) *[where s = current state]*
Let C be the union of the set of plan constraints for E_g and the learned constraints for all $p \in P$
Let U be the overall linear utility function as defined above

Let $\mathbf{X} = \{\}$ *(candidate sets of potentially optimal parameter values)*
For $\beta \in$ *Binding–Sets(D)* do
    $\mathbf{X} = \mathbf{X} \bigcup$ *Optimize*$(P, C\beta, U)$
$\vec{V}$ = *Maximize*$(U, \mathbf{X})$

The *Maximize*$(U, \mathbf{X})$ function returns the $\chi \in \mathbf{X}$ such that the function $U(\chi)$ takes its maximum value. The *Optimize*$(P, C\beta, U)$ function returns a vector of parameter values for the parameter set P subject to the constraints C (with substitutions β performed) such that the utility function $U(P)$ is maximized. The vector $\vec{V} \in$ *ProjectedRegion* produced by the above algorithm gives optimal settings for each of the plan parameters and thus fully determines the plan which will be executed.

The *Optimize* function can be performed by any linear optimization algorithm since constraints are restricted to linear form. We chose the SIMPLEX[10] method. However, methods which handle non-linear optimization are the topic of much current research [Press86 , p. 325.]. Progress in this area can be brought immediately to bear on this implementation of permissive planning by allowing more ready use of nonlinear parameter constraints.

3.4. Refining the Plan

As discussed earlier, every operator in a permissive plan has associated expectations which must hold during execution of the operator. If sensor readings carried out during execution fail to meet these expectations, it is necessary to consider if and how the plan should be refined.

3.4.1. When to refine the plan

Every plan has a target probability of success and coverage *TPSC* and a desired confidence δ just as was prescribed in Chapter 2 on Page 17. A plan is refined when it is known with δ confidence that the probability of success and coverage of the current plan is below the desired *TPSC*. The distribution-independent Nádas stopping criterion is used as in Chapter 2 to decide when sufficient examples have been gathered to conclude whether $(X_i - TPSC) < 0$ with δ confidence where X_i is a measurement of the probability of success and coverage of the plan for one example. The Nádas function and the measurement function for probability of success and coverage are both found in the Appendix.

Beyond the above procedure, a statistical analysis is not performed to evaluate different courses of action during refinement as it was in the algorithm presented in Chapter 2, because we employ a powerful heuristic for determining which of several failures (and thus refinements) to pursue. Al-

10. Details of the SIMPLEX algorithm can be found in [Press86 , pp. 312-326].

though a more rigorous statistical evaluation of the different alternatives could be performed in the implementation, gathering such quantities of experimental evidence is expensive and the heuristics suffice as we will show with our experimental results in Chapters 4 and 5.

3.4.2. Failure hypotheses

Once a decision has been made to refine the plan, it is necessary to find candidate hypotheses for why the plan failed. In permissive planning, all failure hypotheses indicate a single inequality constraint which is considered to have failed and attribute the failure to one of its argument quantities based on an approximation. That is, all failures are considered to be rooted in bad explicit approximations. Let *Approximate*(v_i) hold if v_i is explicitly approximate. Let *Inequality-Constraint(x)* hold if and only if $x \in \{ \leq, \geq, posineq, negineq \}$. Let *Coefficient(i,v)* return the coefficient of variable v in inequality i for predicates *posineq* and *negineq*, return 1 if the variable is the second argument of a $\leq$ predicate or the first argument of a $\geq$ predicate, and return -1 if the variable is the first argument of a $\leq$ predicate or the second argument of a $\geq$ predicate. Failure hypotheses can be expressed as follows:

Let E_g = the generalized explanation structure
Let e = the failing expectation (a node in the explanation structure E_g).

The set of failure hypotheses is $\{<i_1,v_1>, \ldots, <i_n,v_n>\}$ where

$$\exists i_i, v_i, F \text{ such that } Multiple-Leaf-Supports(F, e, E_g) \wedge i_i \in F \wedge Inequality\text{–}Constraint(Predicate(i_i)) \wedge v_i \in Arguments(i_i) \wedge Coefficient(i_i, v_i) <> 0 \wedge Approximate(v_i)$$

These hypotheses can be ranked with regard to the specific failure state using the "distance to failure" of each inequality constraint and the identified argument. Since according to our assumptions regarding errors, small deviations tend to be more common than large deviations, this method of ordering assures that the most likely failure hypotheses are explored first.

3.4.3. Tuning hypotheses

The tuning hypotheses for each failure hypothesis explores all possible ways in which single controllable parameters can be increased or decreased to reduce the chance of the hypothesized failure. It is the responsibility of the user to identify those parameters which the system may use in tuning. If the user allows a parameter p to be used in tuning the predicate *Controllable(p)* holds.

Given a failure hypothesis $<i,v>$, the type of inequality and the argument coefficient prescribe whether v should be increased or decreased. A controllable parameter must then be found which moves v in the desired direction. The tuning hypothesis consists of a pair $<p,m>$ where p is the controllable parameter and m is a mode ($m \in \{increase, decrease\}$).

Tuning hypotheses can be expressed as follows:

Let $<i,v>$ be a failure hypothesis

Let t_m (tuning mode) = increase iff

$$((Predicate(i) \in \{ \leq, posineq\}) \wedge (Coefficient(i, v) > 0)) \vee$$
$$((Predicate(i) \in \{ \geq, negineq\}) \wedge (Coefficient(i, v) < 0))$$

Let t_m (tuning mode) = decrease iff

$$((Predicate(i) \in \{ \geq, negineq\}) \wedge (Coefficient(i, v) > 0)) \vee$$
$$((Predicate(i) \in \{ \leq, posineq\}) \wedge (Coefficient(i, v) < 0))$$

For each failure hypothesis $<i,v>$ the set of tuning hypotheses is

$\{ < p_1, m_1 >, \ldots, < p_n, m_n > \}$ where

$\exists p_i, m_i$ *such that*

$$(t_m = increase \wedge Controllable(p_i) \wedge Q+(p_i, v) \wedge m_i = increase) \vee$$
$$(t_m = increase \wedge Controllable(p_i) \wedge Q-(p_i, v) \wedge m_i = decrease) \vee$$
$$(t_m = decrease \wedge Controllable(p_i) \wedge Q+(p_i, v) \wedge m_i = decrease) \vee$$
$$(t_m = decrease \wedge Controllable(p_i) \wedge Q-(p_i, v) \wedge m_i = increase)$$

Here, the predicate *Q+(a,b)* signifies that an increase in the quantity b results in an increase in the quantity a and a decrease in the quantity b results in a decrease in the quantity a. The predicate *Q-(a,b)* similarly signifies an inverse relationship between the two argument quantities. Since the

relationships between quantities come from the explanation of the permissive plan, it is possible to compute the exact relationship between any two such quantities.

Each tuning hypothesis seeks to decrease the chance of the hypothesized failure. The plan constraints supporting the failed expectation can affect its probability of success when their arguments are explicit approximations. The tuning hypothesis seeks to increase the probability that the hypothesized failing inequality will succeed. If a controllable parameter and direction exist which can successfully accomplish this, it is guaranteed to be included in the set of tuning hypotheses.

3.4.4. An algorithm for permissive plan refinement

Let $ETR \subseteq ProjectedRegion \subseteq S'_{GP}$ be a region defined by the conjunction of the plan constraints and the learned constraints which impose lower and upper bounds on parameter values. Let $\vec{P} \in ETR$ be a vector of parameter values. Let $L_{i,ETR}(\vec{P})$ and $U_{i,ETR}(\vec{P})$ be linear functions which give the lower and upper bounds on the value of parameter P_i as a function of the other parameter values P_j with $j \neq i$. These lower and upper limits may be changed during refinement but are initially set to reflect the plan constraints. Let $Util_{i,ETR}(P_i)$ be a piecewise linear utility function with one or two pieces. If $Util_{i,ETR}(P_i)$ has two pieces, they are noted $Util1_{i,ETR}(P_i)$ and $Util2_{i,ETR}(P_i)$. $Util_{i,ETR}(P_i)$ is initially set to a constant value for all dimensions i to give a completely flat utility function over the *ETR* before any refinement has taken place.

The refinement algorithm takes as input a tuning hypothesis $<P_i, mode>$ where P_i is a particular plan parameter to be tuned and *mode* is either *increasing* or *decreasing* to indicate the direction to tune. Recall that a statistical technique has been used prior to our refinement algorithm to decide if it is necessary to perform refinement to achieve the target success rate for the plan. The first step in the refinement algorithm is to check if the size of ETR is below a small fixed threshold where further refinement would make the plan very unlikely to apply. If ETR is still of sufficient size, refinement continues. The result of refinement may be to change either the learned constraints (by changing

$L_{i,ETR}(\vec{P})$, $U_{i,ETR}(\vec{P})$, and/or $Util_{i,ETR}(P_i)$) or one or more of the weights w_i in the overall linear utility function. How the learned constraints are changed depends on the tuning hypothesis, the value of P_i for the failure, and the current state of the learned constraints. One of the following five cases will apply, four requiring an update of the learned constraints and one requiring a change in weights:

Case 1:

If $Util_{i,ETR}(P_i)$ is constant and the tuning hypothesis is $< P_i, increasing >$ then $Util_{i,ETR}(P_i)$ is given a positive slope as shown in Figure 3.7.

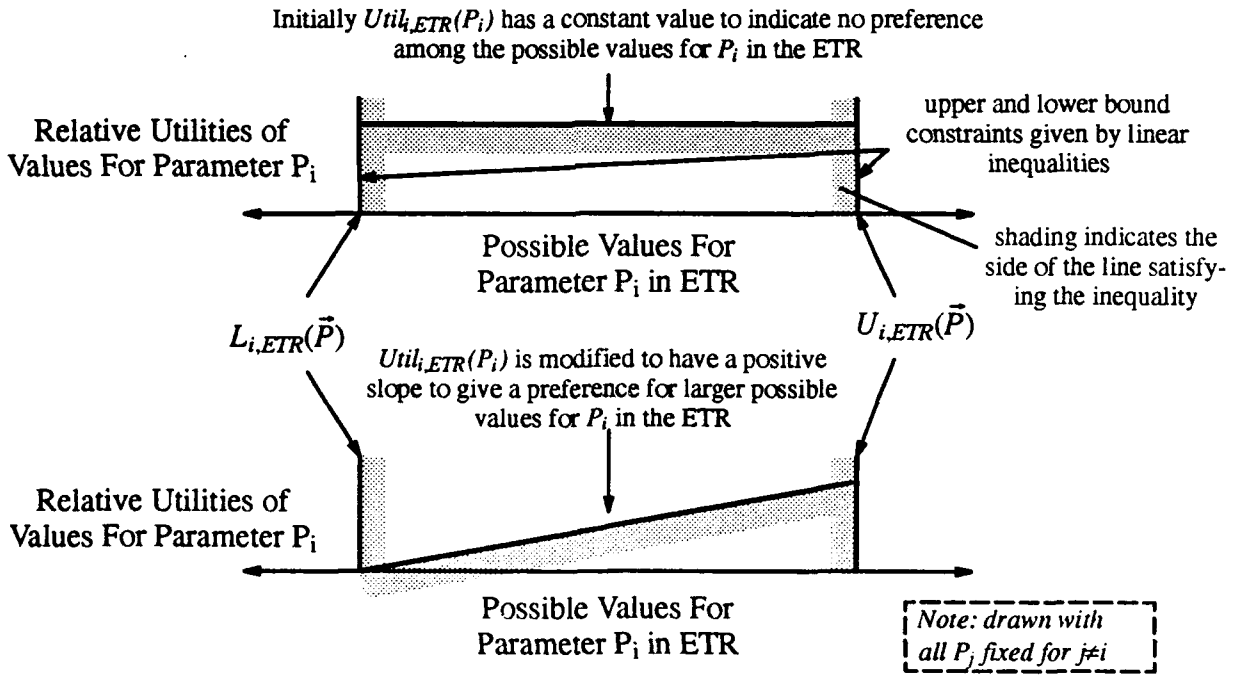


Figure 3.7. Refinement Case 1

Case 2:

If $Util_{i,ETR}(P_i)$ is constant and the tuning hypothesis is $< P_i, decreasing >$ then $Util_{i,ETR}(P_i)$ is given a negative slope as shown in Figure 3.8.

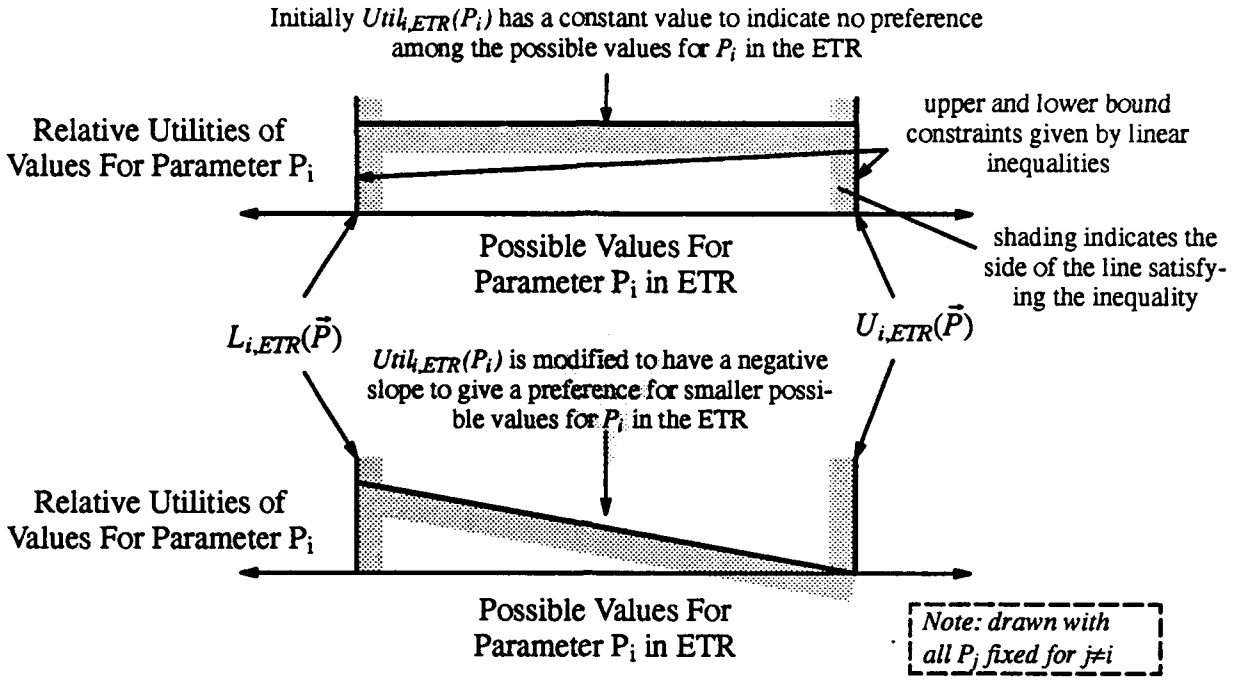


Figure 3.8. Refinement Case 2

Case 3:

This case applies if the tuning hypothesis is $< P_i, increasing >$ and *increasing* is not *consistent* with the piecewise linear function $Util_{i,ETR}(P_i)$. Here, if the piecewise function at the value for P_i for the failure is *decreasing* it is not consistent. The lower bound $L_{i,ETR}(\vec{P})$ is modified by adding a constant to yield a value of P_i where P_i is the value of the parameter for the failure. A peak for the piecewise linear function is chosen halfway between the new $L_{i,ETR}(\vec{P})$ and $U_{i,ETR}(\vec{P})$. $Util1_{i,ETR}(P_i)$ and $Util2_{i,ETR}(P_i)$ are then assigned positive and negative slopes, respectively, with their intersection occurring at the peak value as shown in Figure 3.9.[11]

11. This refinement is consistent with that presented in Chapter 2 in that ETR is being increasingly constrained to eliminate failure. However, decisions such as adding a constant to the lower bound or placing the peak of the utility function exactly between the two bounds are decisions made in this implementation from many possible schemes which fit the basic requirement of permissive planning.

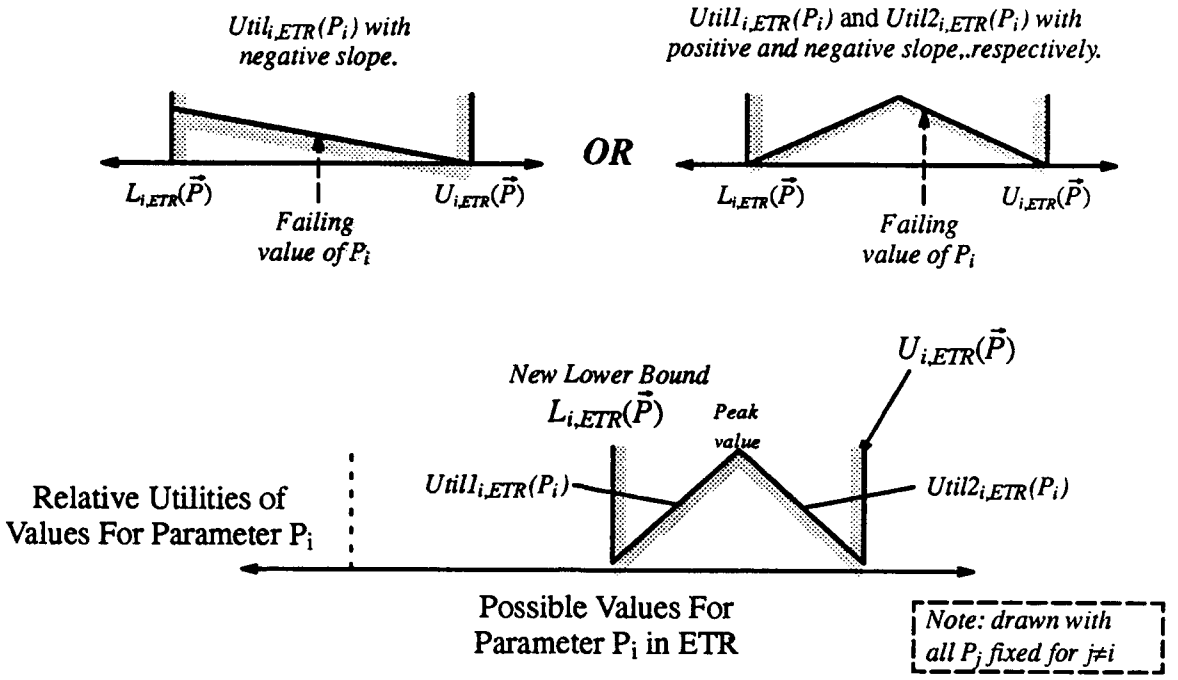


Figure 3.9. Refinement Case 3

Case 4:

This case applies if the tuning hypothesis is $< P_i, decreasing >$ and *decreasing* is not *consistent* with the piecewise linear function $Util_{i,ETR}(P_i)$. Here, if the piecewise function at the value for P_i for the failure is *increasing* it is not consistent. The upper bound $U_{i,ETR}(\vec{P})$ is modified by subtracting a constant to yield a value of P_i where P_i is the value of the parameter for the failure. A peak for the piecewise linear function is chosen halfway between $L_{i,ETR}(\vec{P})$ and the new $U_{i,ETR}(\vec{P})$. $Util1_{i,ETR}(P_i)$ and $Util2_{i,ETR}(P_i)$ are then assigned positive and negative slopes, respectively, with their intersection occurring at the peak value as shown in Figure 3.10.

Case 5:

This case applies if the tuning hypotheses is consistent with the piecewise linear function $Util_{i,ETR}(P_i)$ as shown in Figure 3.11. It must be the case that a utility preference constraint for

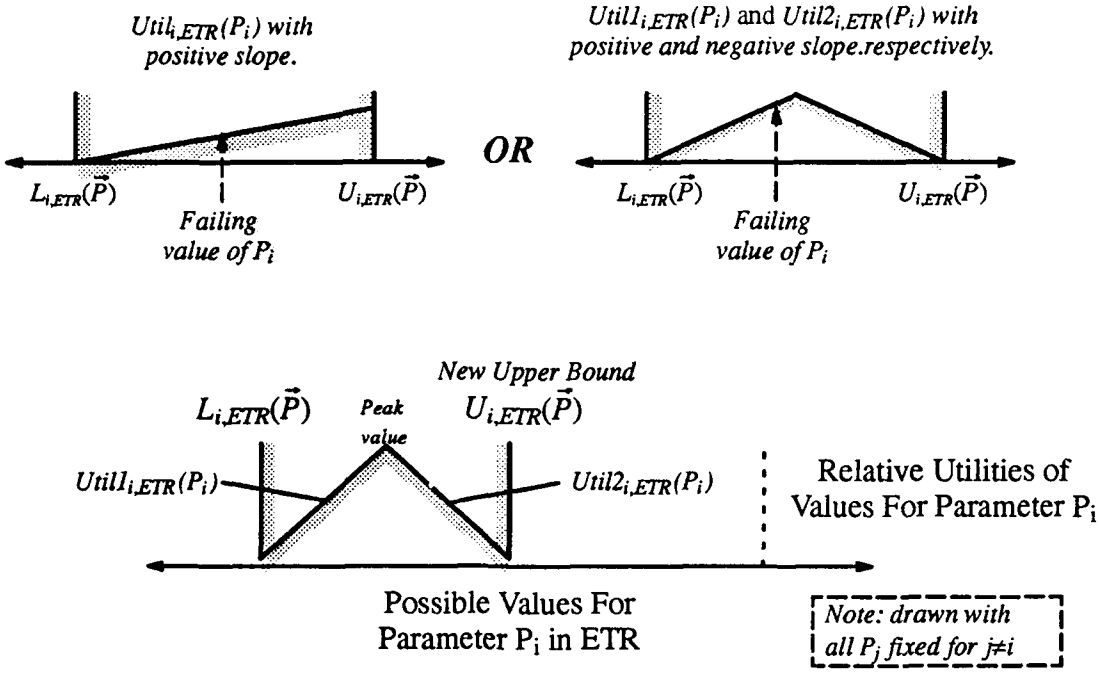


Figure 3.10. Refinement Case 4

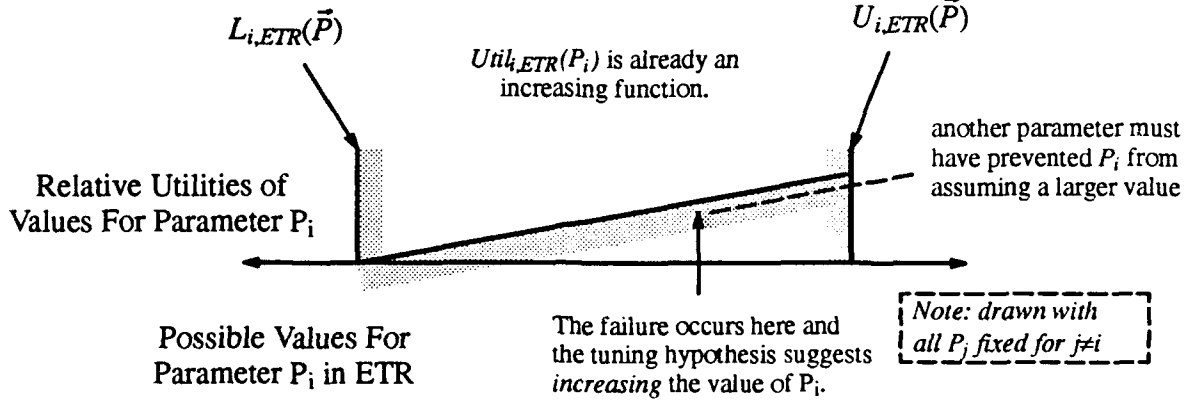


Figure 3.11. An Instance of Refinement Case 5

another parameter or parameters with weights greater than or equal to the weight for parameter P_i compete with the preferences for setting P_i. If this is not the case, then ETR can be modified no further in the way the tuning hypothesis suggests. The weights for the parameters must be adjusted such that the weight for P_i is greater than the weight for the competing parameter(s). Every requested weight adjustment can be viewed as imposing more constraints on the weights. If it ever becomes

the case that the necessary new weight constraints cannot be satisfied while maintaining previous weight constraints, the tuning hypothesis can no longer be implemented with the plan.

We assume, without loss of generality, that each weight w_i is subject to the initial constraints $\epsilon \leq w_i \leq 1$ where ε is a very small positive real number. We use ε to avoid the degenerate weight solution of all zero-valued weights.

Recall that parameter values are selected by choosing values for any underconstrained continuous parameters for each of the finite sets of binding sets for the discrete leaf supports. These are then combined by evaluating the result for each of the binding sets using the same scheme. In fact, because preferences are linear and are combined by a linear global utility function, there exists a discrete set of parameter value vectors, one for each binding set and set of extreme values within the set. One of the elements of the parameter value vector set gives the optimal value of the global utility function with respect to continuous parameters for each possible set of parameter weights. When some specific value for the weights is given, the effect of calling the optimization function is to produce the parameter value vector from this set giving the highest utility. In reasoning about a failure in choice of relative weights between parameters, the system seeks to influence which of the possible parameter value vectors is chosen.

Let *PVVS* be the parameter value vector set discussed above. Let C_w be the current set of constraints on values assigned to the weights w_i associated with each parameter P_i at the time of the failure. Let $<P_i, mode>$ be the tuning hypothesis. Let $Util_{i,ETR}$ be the piecewise linear utility function (combining $Util1_{i,ETR}$ and $Util2_{i,ETR}$). The weight constraint update algorithm, given below, returns weight constraints (if any) which can be added to prevent the encountered failure.

Let NewConstraintSets = {}
Foreach $PS \in PVVS$ **Do**
Begin
If (*moving from value* P_i *to value* PS_i *(at the time of failure) satisfies the direction of tuning indicated in the tuning hypothesis)* $\wedge$

$$\textit{Satisfiable}\left(C_w \wedge \sum_{i=1}^{n} \left(Util_{i,ETR}(PS_i) - Util_{i,ETR}(P_i)\right)w_i \geq \epsilon\right) \textbf{ Then}$$

$$NewConstraintSets = NewConstraintSets \cup \left\{ C_w \cup \sum_{i=1}^{n} \left(Util_{i,ETR}(PS_i) - Util_{i,ETR}(P_i)\right)w_i \geq \epsilon \right\}$$

End

The ***Satisfiable*** predicate in the above algorithm holds if and only if a solution for the weights w_i exists satisfying all constraints in C_w as well as the additional linear constraint. Epsilon is a very small positive real number which ensures that the candidate parameter value set which satisfies the tuning direction of the tuning hypothesis will be selected over the current parameter value set.

The procedure maintains a search space of possible sets of constraints on weights for the plan in the current state of parameter constraints. If the constraint update algorithm is unable to produce a new consistent constraint set, a previously unexplored constraint set is made active. The current active set of weight constraints is used in the selection of a set of weights for plan application.

3.5. Maintaining a Plan Library

We have seen how refinement of a single permissive plan proceeds. However, a permissive planning system must maintain a library of permissive plans in various states of refinement. In order to facilitate testing the application of permissive plans and because of substantial overlap in preconditions between differently refined variants of a base plan, it is efficient to index plans in a plan application tree. Every link in the tree corresponds to one or more plan preconditions which must be satisfied to traverse the link. An ordered set of 0 or more plans is attached to each tree node. The application tree is traversed in depth-first fashion. If a node is reached through application of the preconditions at links above it, the plans at the node are applicable.

A permissive planning system is not guaranteed to be able to find a working plan for every goal and situation. If any of the links in the plan application tree can be traversed, yet no applicable plan is found, then the permissive planner is unable to generate a working plan given the domain theory, goal, world state, and plan success criteria defined for the system. This is true because all possible initial general plans for a goal are indexed as well as all possible plan refinements for encountered failures. Plans may be removed from the tree if they fail and further refinement is not possible.

If no links in the application tree were successfully traversed, then a set of explanations is generated for how the goal may be achieved from the current state. Those explanations are generalized and packaged into general plans which are then indexed in the plan application tree.

In plan refinement, a set of failure hypotheses are generated based on a failed expectation. A set of tuning hypotheses is generated for each failure hypothesis. Each tuning/failure hypothesis pair is considered (in the order of failure plausibility discussed earlier). If the refinement cannot be made to the plan (e.g., the plan is sufficiently constrained with regard to the parameter in question that no further tuning would be of use), the tuning/failure pair is rejected. If the refinement can be made, a copy of the plan is made and refined. If no learned bound constraints are changed in refinement, and hence the preconditions have not changed, the new plan is indexed at the same node as the current plan. If a bound constraint is changed, a corresponding precondition is added via a link extending below the node where the current plan is indexed and the new plan is attached to a new node below this link. Once all of the tuning/failure hypothesis pairs have been processed, the original plan is removed from the tree (now replaced by its tuned variants). Plans at a node and sibling nodes in the tree are always ordered by plausibility of the failures which gave rise to them.

Our implementation of permissive planning described here mirrors the theoretical account given in Chapter 2 but with several important differences. The implementation has been modulated by a need to work with a concrete explanation structure supporting the plan. Consequently, using that explana-

tion structure, parameters could be identified, which led to our approximated space S'_{GP}. Rather than limit ourselves to axis-aligned constraints, we sought more expressive power in the implementation by allowing general linear constraints. Learned utility constraints were introduced to help guide selection of parameters within the ETR. To reduce the number of examples needed prior to tuning, a statistical evaluation is only employed to determine if to refine, not how to refine. In the next two chapters we introduce two domains in which the implementation was employed and give empirical results of the approach.

4. PERMISSIVE PLANNING IN THE GRASPING DOMAIN

In order to demonstrate and test our approach, we chose a complex real-world domain where uncertainty plays a role: robotic grasping. The goal in this domain is to learn to control a robotic manipulator to successfully grasp objects in its workspace. Planning of grasps for arbitrarily shaped objects is still an active research problem, as evidenced by the number of related papers presented at the 1992 IEEE International Conference on Robotics and Automation. Uncertainty is one primary difficulty in this domain. Sensors do not return precise information. Visual sensors seeking to identify or help represent the objects are very sensitive to placement of the light source. For example, an observer blocking some of the light may actually be affecting visual sensing of the objects. Force sensors used on the manipulator also are subject to errors so that the precise position at which the manipulator first contacts an object is not known exactly. The robotic manipulator also cannot be completely precise in its movement. The system must represent the world in order to construct plans for carrying out its actions. For example, in using visual sensors to model an object or in recognizing objects and retrieving a pre-stored model, that model exists at some resolution. The greater the resolution, the more information that must be considered in planning to grasp the object. Therefore, in order to allow plans to be constructed in a reasonable amount of time, object models must be simplified. Altogether, the robotic grasping domain provides a challenging testbed for learning techniques.

Figure 4.1 shows the laboratory setup. Our implementation of permissive planning is called GRASPER and is written in Common Lisp running on an IBM RT125. GRASPER is interfaced with a frame grabber connected to a camera mounted over the workspace. The camera produces bitmaps from which object contours are extracted by the system. The system also controls an RTX SCARA-type robotic manipulator. The RTX has encoders on all of its joint motors and the capability to control many parameters of the motor controllers including duty cycle (of motor current). This gives the system a rudimentary capability of detecting collisions with the RTX gripper. If a large enough duty cycle is used with the motor to overcome the friction of the joint and the position encod-

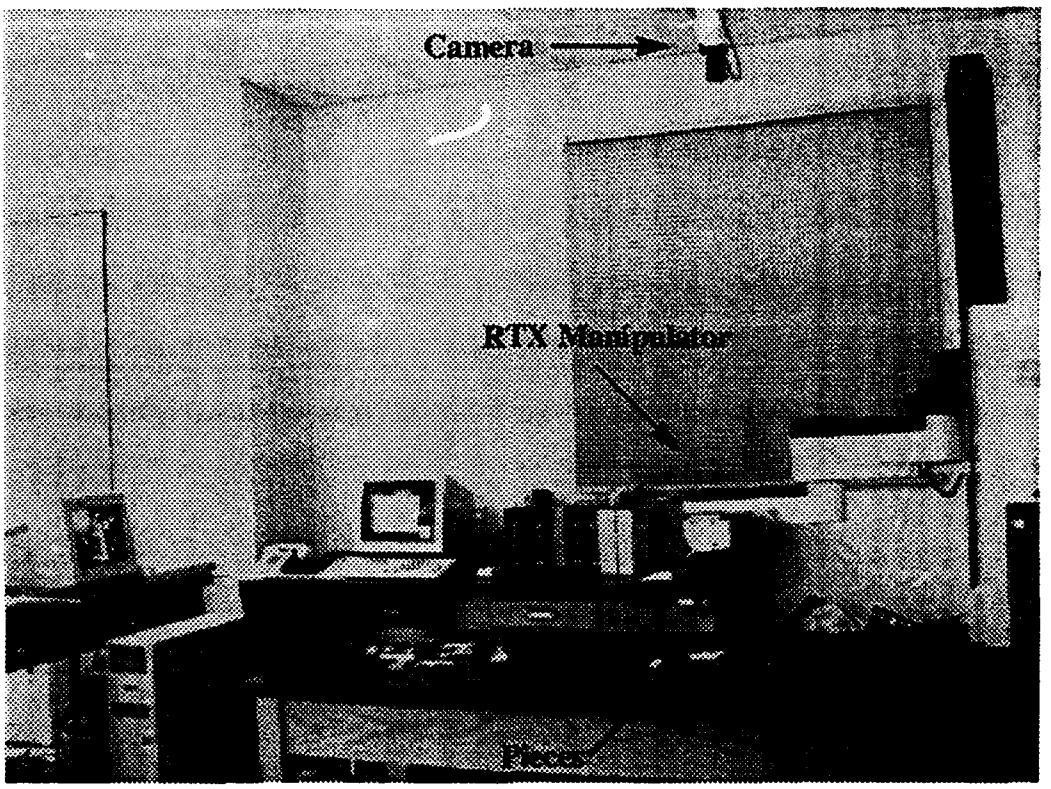


Figure 4.1. GRASPER Experimental Setup

er indicates no movement, an obstacle has been encountered. This type of sensing gives rudimentary force feedback during execution of a plan. Such feedback is important for carrying out monitored actions in the world.

One test of the GRASPER system in the robotics grasping domain is to successfully grasp the plastic pieces from puzzles designed for young children. Since the pieces are laminar (relatively flat and of fairly uniform thickness), an overhead camera is used to sense piece contours. The GRASPER system could potentially be extended to work with three-dimensional objects given sufficient hardware to return three-dimensional object information. These laminar pieces we use have interesting shapes and are large enough, yet challenging, to grasp. The goal is to demonstrate improving performance at the grasping task over time in response to failures. The failures that the current implementation learns to overcome, when using isolated grasp targets, include learning to open wider to avoid stubbing the fingers on an object, learning to prefer more parallel grasping faces to prevent

unstable grasps, learning to grip with more force to keep the object from twisting out of grasp, and learning to grasp near the center of mass of the object to minimize twisting. Detailed examples of three of these failures follow later in this chapter.

4.1. Execution and Monitoring for Robotics

Every operator employed in the plan has a set of associated sensor expectations expected to hold during and after its execution. Sensors can be monitored during execution of a plan to see if their actual execution trace deviates from the expected profile. If the expectations are violated, failure refinement can begin. The expectations also describe acceptable bounds on sensor readings at termination of the operation.

A *profile* for a set of sensors is a series of one or more partial sensor-based state specifications. First, let us consider how such sensor-based states can be specified. Suppose that we are slowly closing a robotic gripper on an object which is known to be between the gripper fingers. Further suppose that the two independent sensors we are interested in are the opening width of the gripper and the force which resists closure of the gripper. The partial state specification for the case in which the gripper still has not contacted the object can be represented by intervals on the values of the two sensors as shown in Figure 4.2:

Opening Width: [10.5,100]	*(width of object is 10.5, maximum opening is 100)*
Resisting Force: [0,5]	*(5 is a small threshold above which the gripper fingers must both have made contact with the object)*

This region is represented by the cross-hatched rectangle at the bottom of the diagram. Also shown is the vertical rectangle representing the expected partial state as the object is squeezed more tightly to establish a strong contact between the gripper and the target object. This is shown by the shaded rectangle oriented vertically. The rectangle has a small width because of flexing that takes place as force is applied. A failure during squeezing, such as the object slipping away, would be repre-

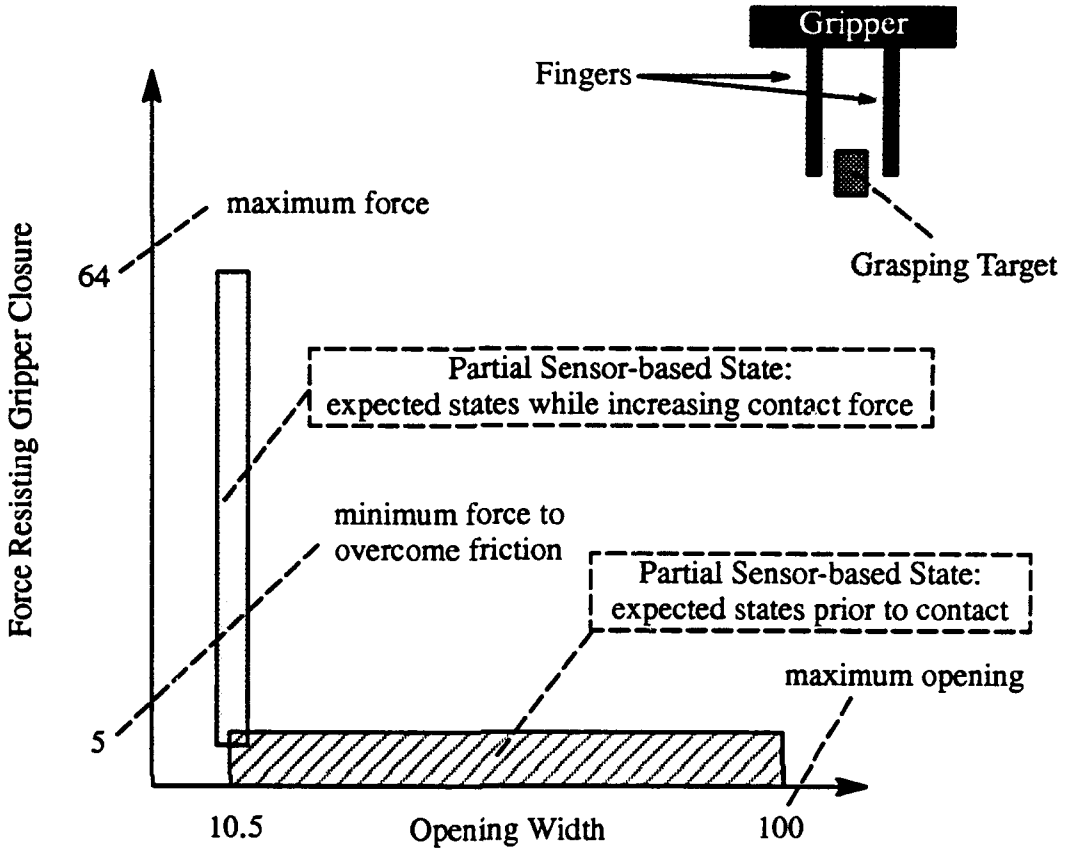


Figure 4.2. Two Sensor-based Partial States

sented by a departure from this expected rectangle. The departure would be to the left of the rectangle eventually ending close to the point (0,64) where the gripper has closed strongly on itself.

Figure 4.3 shows a profile for squeezing the gripper on a piece which consists of a series of the two sensor-based states of Figure 4.2. The figure also illustrates one possible sensor trace which adheres to the profile and thus satisfies the expectations for the squeezing operation.

The profiles just described are an important part of the system's monitoring capability. It is important that the system be able to represent the actions to be carried out, the expected outcomes of those actions, and the justifications for those outcomes. In the robotic grasping domain, the set of actions

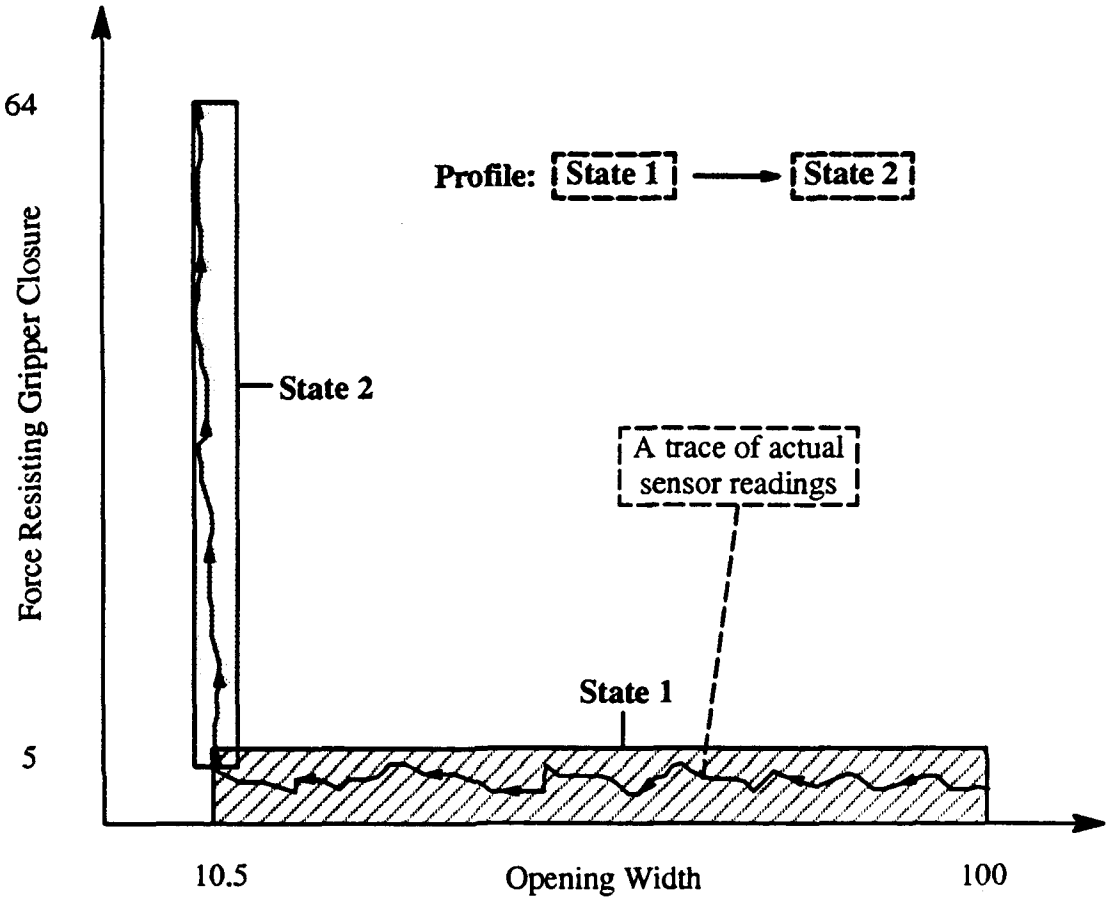


Figure 4.3. A Profile for Squeezing a Piece

to be monitored is a set of motor commands to the manipulator. These may occur as individual motor moves as with a command to move the arm up the column in the case of our SCARA-type manipulator. A group of simultaneously applied primitives may also be monitored. For instance, in moving the manipulator while grasping an object, force has to be continually applied to squeeze the object while the other joints' moves are being carried out. In this case, the expectation profile may apply both to whether the object is sensed between the fingers and whether an external force is sensed by the arm during the motion.

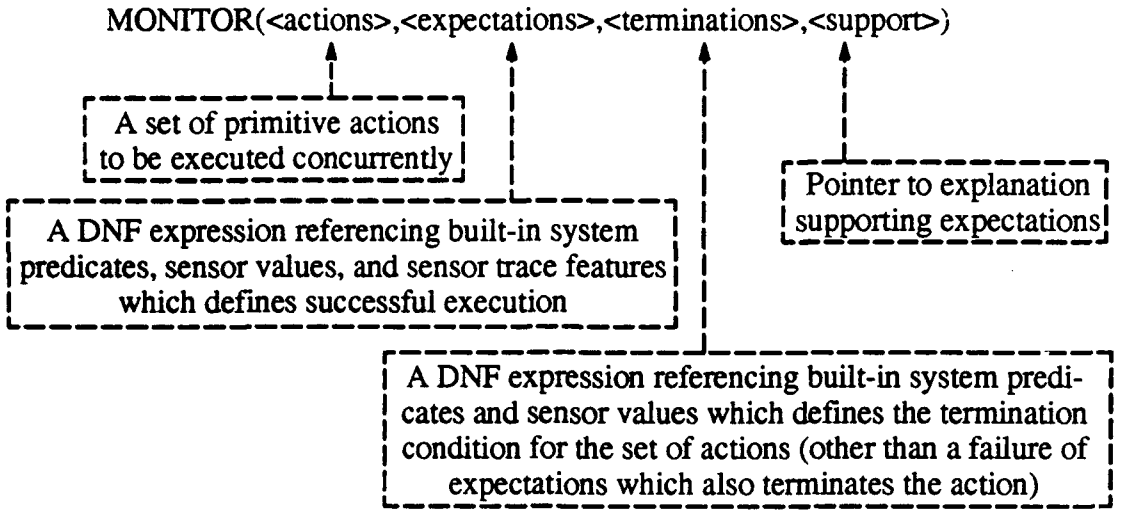


Figure 4.4. Syntax for Monitored Actions

Figure 4.4 illustrates the syntax of monitors in the grasping implementation. The monitor specifies one or more coordinated actions which are performed simultaneously. Expectations are specified which are evaluated continually during execution, in the case of sensor expectations, and are also checked after termination of the action, in the case of expected features of a full sensor trace. Terminations specify under which normal conditions the set of actions should be halted. The action is also terminated if the expectations fail to hold during execution. Any monitored set of actions employed in a plan must have its expectations justified. The support field of a monitor specifies a predicate which, if proven, justifies that the expectations will hold throughout execution of the monitor.

Suppose that we wished to monitor the gripper position and detected a force while attempting to surround an object. Let us assume the expected profile is that shown in Figure 4.2 on Page 51. This profile is justified by an explanation supporting why the grasp chosen is a stable one. Figure 4.5 shows a monitored action that satisfies these conditions. The *Move-Gripper* primitive action is specified and the direction of movement is to close the gripper (this monitored action would be used when the gripper already surrounds the object). The expression for the expectations ensures that during execution the force and position of the gripper lie in one of the two rectangles defining the

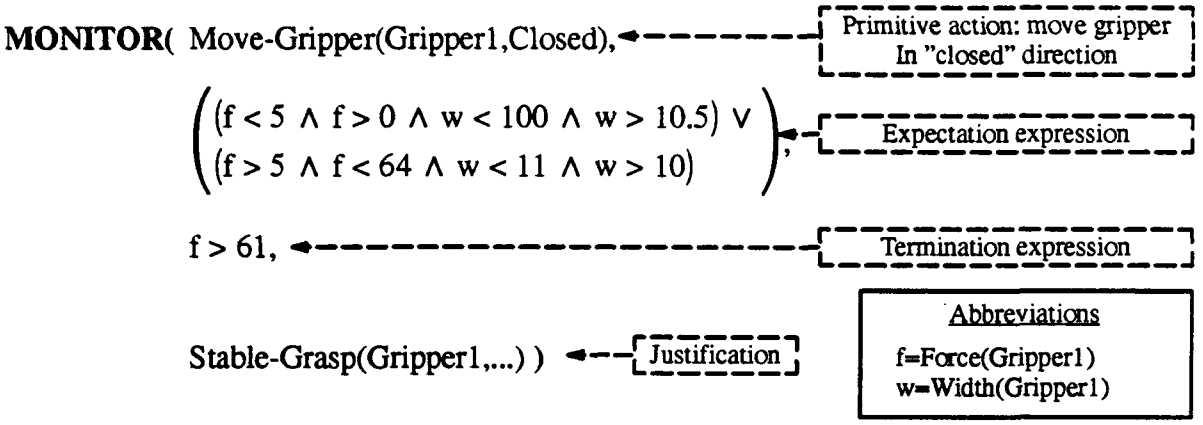


Figure 4.5. An Example of a Monitored Action

profile as shown in Figure 4.2 on Page 51. It also ensures that when execution has stopped that the final resulting force should exceed 60 units and the final width should be greater than 10 units. The expression defining the expectations terminates the action if the force and position ever leaves both rectangles defining the profile. It also terminates the action if the force exceeds 61 units and lies in the vertical rectangle. This is the expected termination. The support for the monitored expectations is justified by a proof that the planned grasp is stable.

In our above example, the expectations and terminations are expressed using the current force and opening width of the gripper during execution. The expectations are also based on an expected final reading for the force and position at the time the action terminates. In general, the expectations and terminations may reference predicates known to the system as well as sensors available on the manipulator. For actual execution on the robot, the sequence of monitored actions specified by a plan is compiled into a Common Lisp program for rapidly checking the sensors while the actions proceed. Many tradeoffs exist in the monitoring process. For instance, since sensors take time to read, the faster the actions are carried out, the smaller is the number of sensor readings which can be obtained. Furthermore, one might read more types of sensors during execution but each will have a lesser number of readings because of the time constraint. However, because of the permissive planning approach, the plans are improved in spite of these factors.

4.2. Two Detailed Learning Episodes in the Piece Grasping Domain

We will first present two experiments with system grasp performance before giving overall empirical results on a larger collection of objects.

4.2.1. Experiment 1

Our first experiment will involve grasping plastic puzzle pieces. These are the same pieces and theory which are used in empirical testing for the grasping domain.

4.2.1.1. Example 1

Figure 4.6 shows the system's status display during a grasping task. First, the system uses the camera to acquire contour information about objects in the workspace. These contours are shown in the upper-left corner of the figure. Next, the contours are approximated with n-gons which result in $\frac{1}{2}\left(n^2 - n\right)$ possible unique grasping face pairs for each object. These approximated object contours appear in the upper-right quadrant of Figure 4.6. The algorithm chooses a value for n such that an approximation to the object is possible within a certain error threshold. The approximated object representations as well as the current information about the state of the robot manipulator are asserted in the initial situation from which the system will apply the plan. The target object is then selected and an explanation is generated for how to achieve a grasp of the target (if no current permissive plan applies). Figure 4.7 highlights the selected target object. The heavy line indicates the approximation to the object contour while the lighter pixels show the actual sensed object contour points. The arrows indicate the positions of the leading edges of the fingers for the grasp position chosen. The explanation supporting this grasp position involves a total of about 300 nodes with a maximum depth of 10 levels.

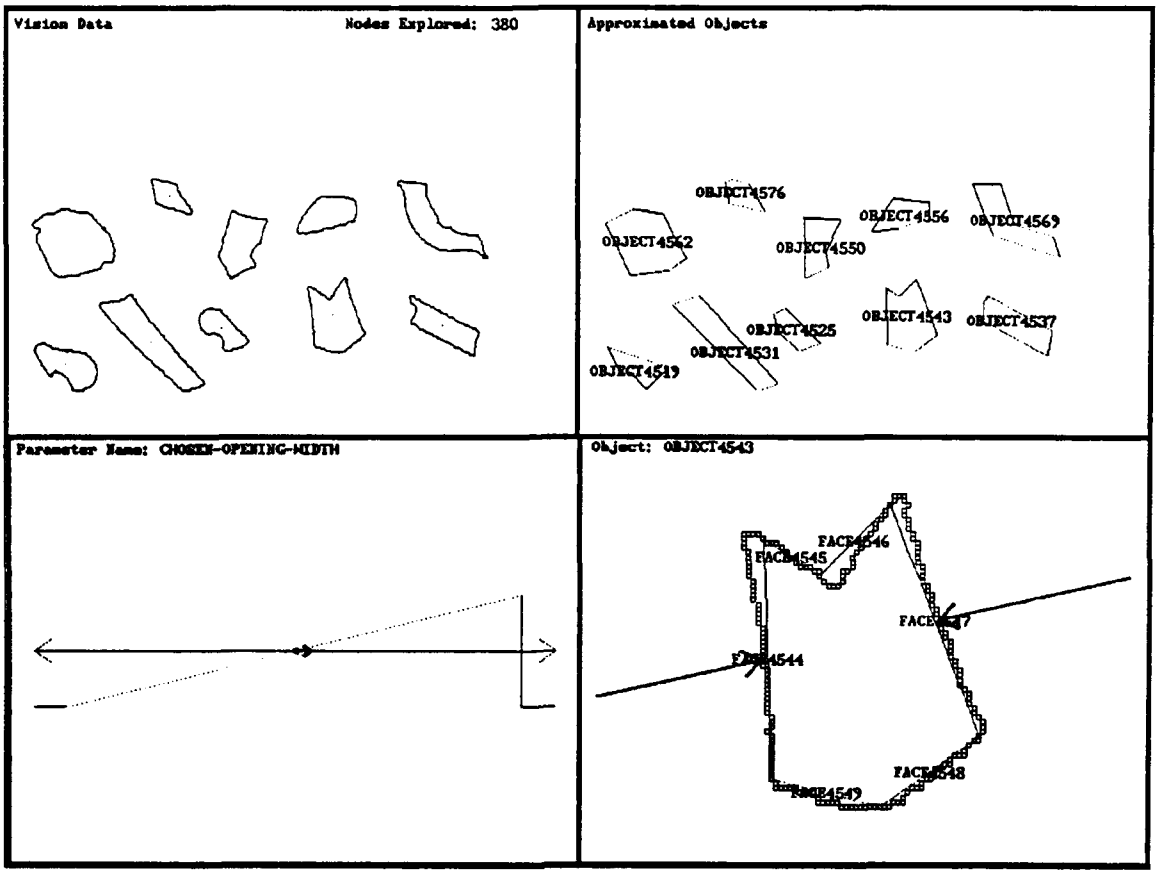


Figure 4.6. System Status Display During Grasp of Object4543

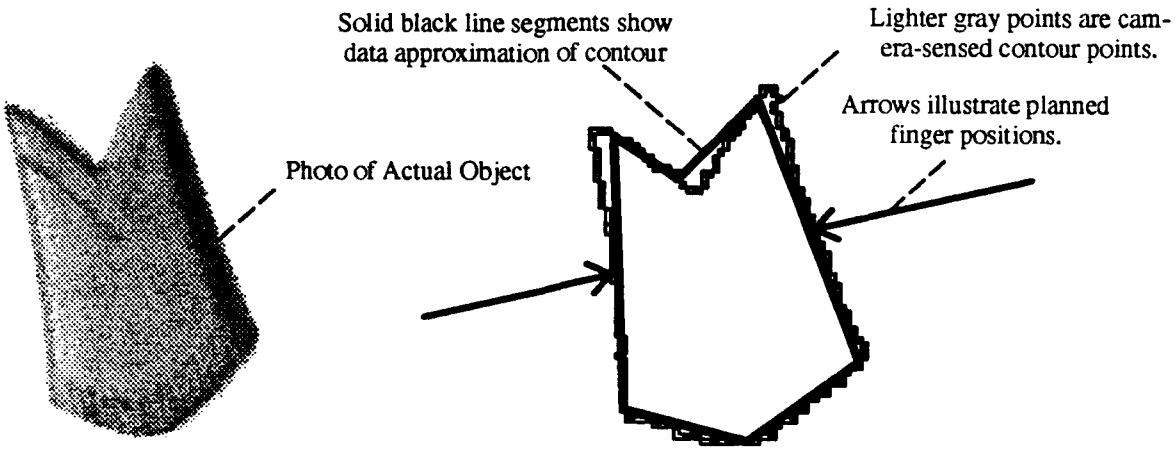


Figure 4.7. Grasp Target and Planned Finger Positions

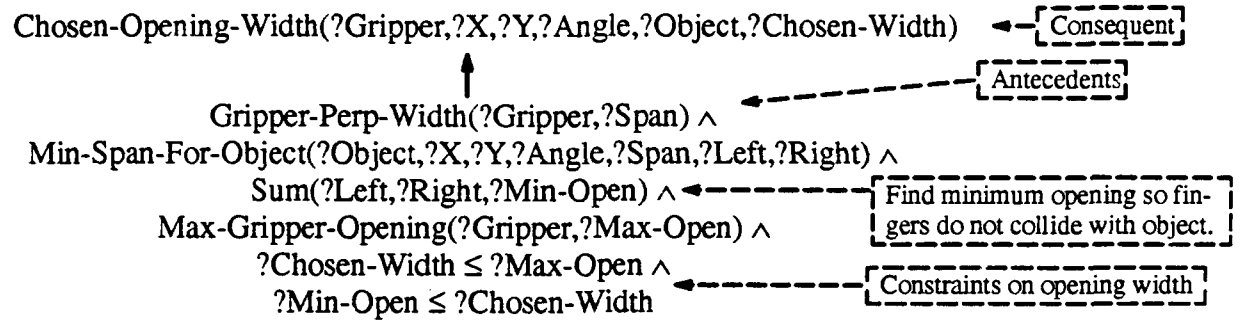


Figure 4.8. A Rule Showing Some Constraints on Opening Width

One of the rules employed in the explanation is shown in Figure 4.8. This rule is employed to constrain the chosen opening width of the gripper to be wide enough to surround the piece but no wider than the maximum opening width for the gripper. Notice that the last two antecedents of the rule are inequality constraints.[12] The value for ?Chosen-Width in this rule, which corresponds to the chosen opening width for the gripper, is underconstrained and qualifies as a tunable parameter. In this case, initially we have a constant utility function for the plan's opening width parameter. A movement of the gripper from its initially closed position to the minimum necessary opening width satisfies the constraints in this case. That opening is depicted by the separation of the arrows in Figure 4.7. After the explanation is generated, and its associated operator sequence executed, the monitored action shown in Figure 4.9 encounters a violation of the expected sensor readings. Figures 4.10 and

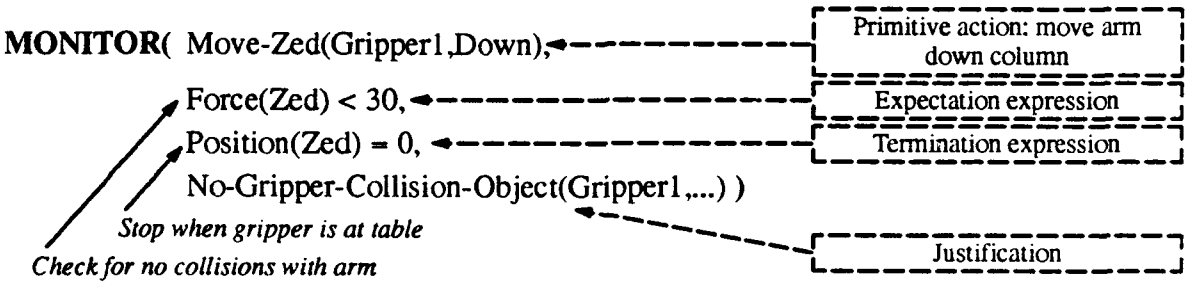


Figure 4.9. The Failing Monitored Action

12. Simple *posineq* and *negineq* linear inequality constraints will be shown in their infix form in this chapter for clarity.

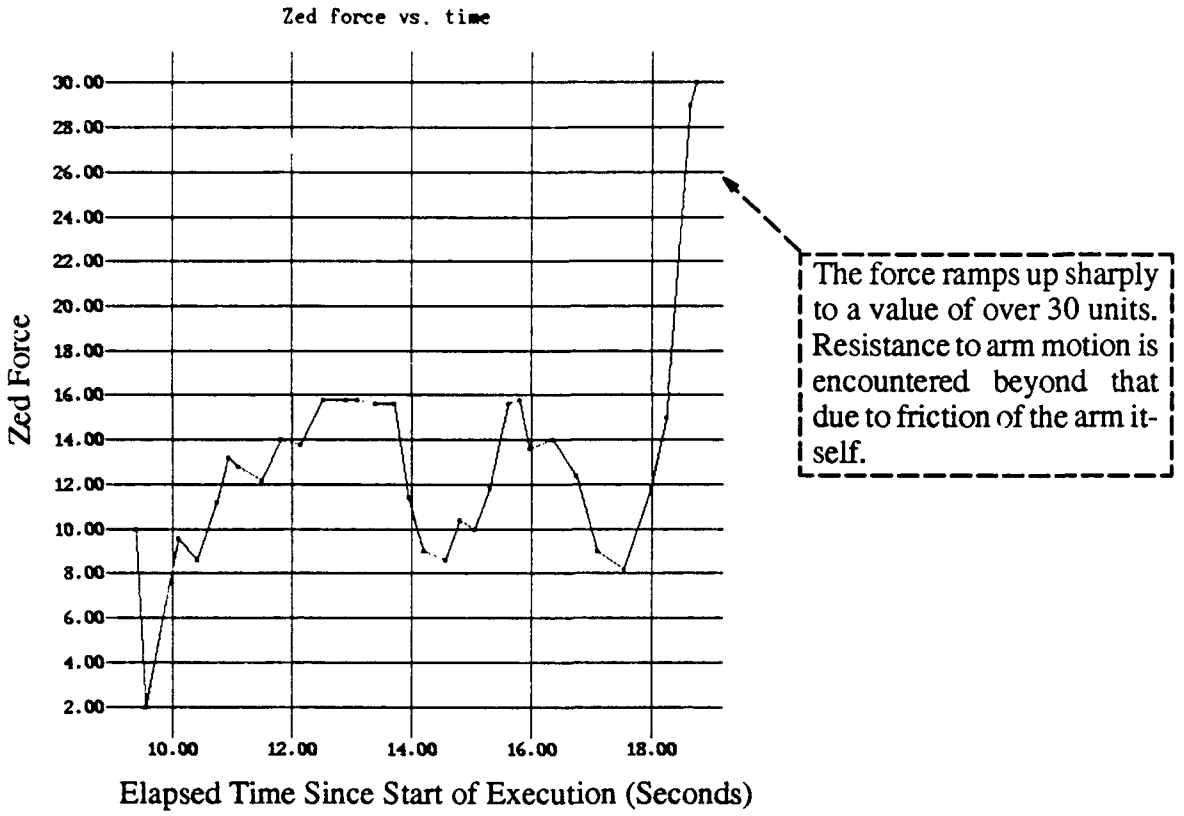


Figure 4.10. Zed Force vs. Time into Action Sequence

4.11 show traces of force and position, respectively, plotted against time (in seconds) into the overall operator sequence.

The original explanation supporting **No-Gripper-Collision-Object** in the above monitored action is now suspect due to the violated expectations. A sketch of the specific explanation is shown in Figure 4.12. This explanation for why no external force should have been sensed during the downward move of the gripper is the starting point for developing the tuning hypotheses. Explicitly approximate quantities and tunable parameters employed in the explanation are identified. In particular, the constraint **?Min-Open** ≤ **?Chosen-Width** on opening width (from the rule in Figure 4.8 on Page 57) is in the set of *Multiple-Leaf-Supports* as described in Chapter 3. The quantity **?Min-Open**, derived from the sensed width of the object at the point of the grasp, is an explicit approximation. The quantity **?Chosen-Width** is a tunable parameter. Therefore, the inequality constraint

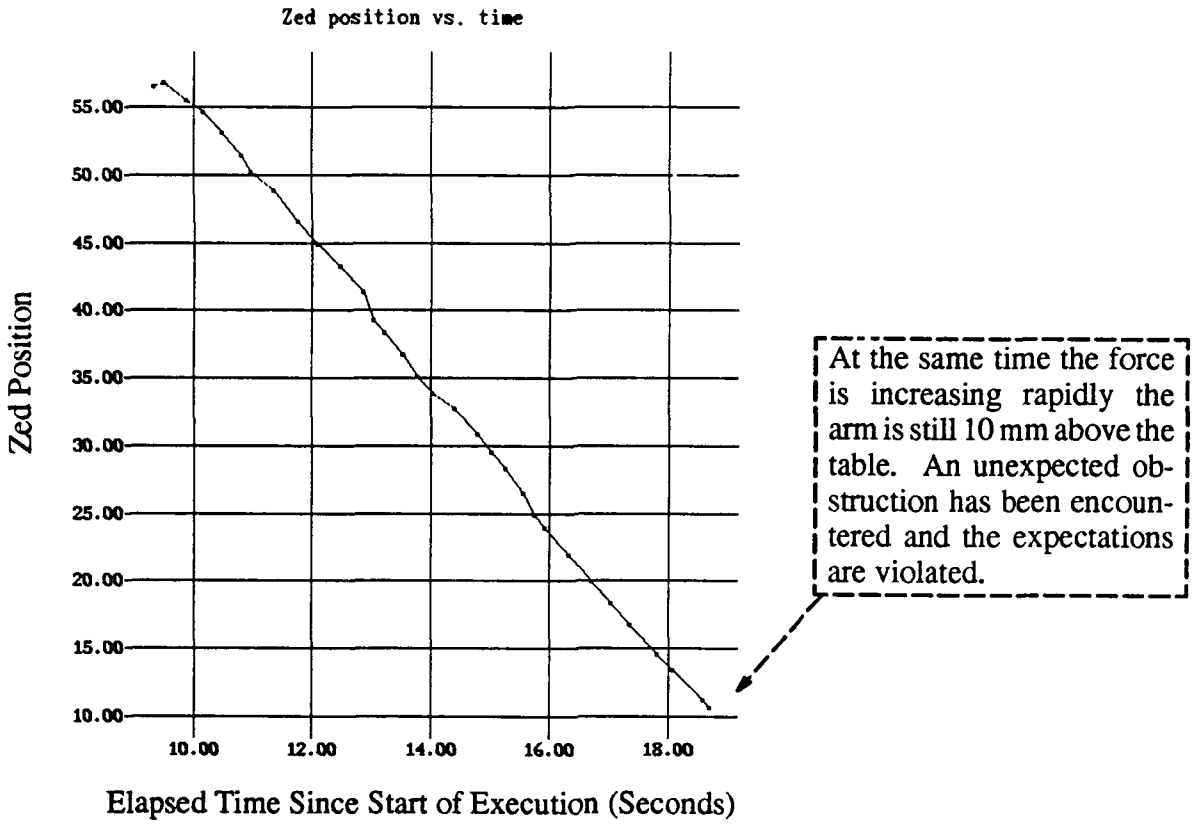


Figure 4.11. Zed Position vs. Time into Action Sequence

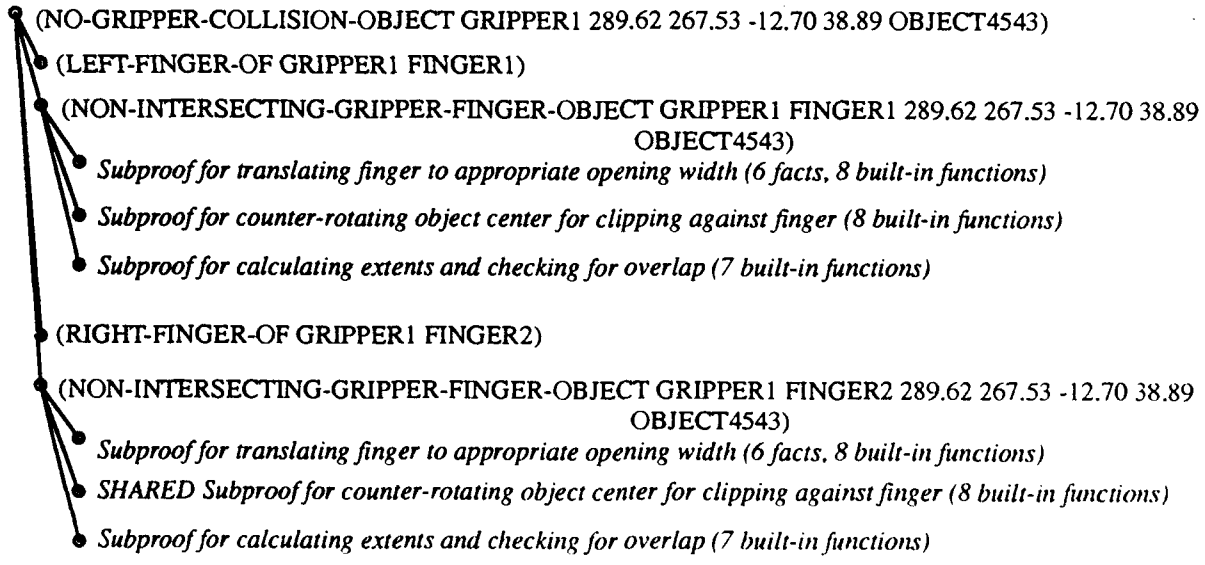


Figure 4.12. Explanation Specific to Failure

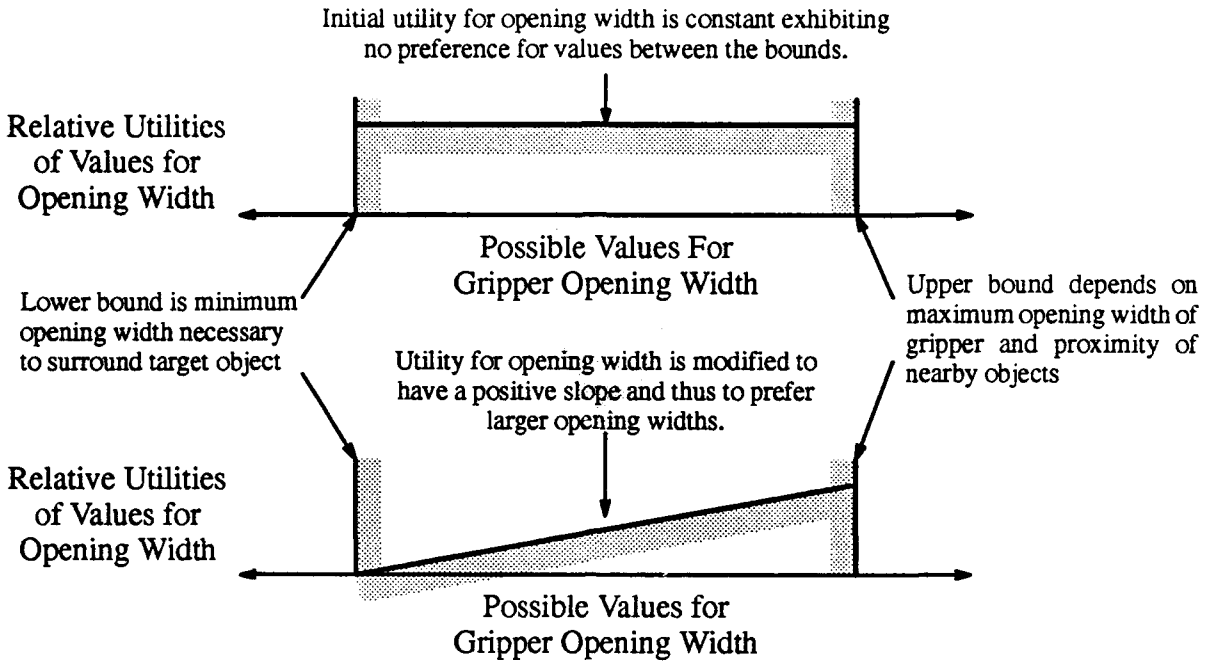


Figure 4.13. The Chosen-Opening-Width Parameter Utility Function Before and After Tuning

?Min-Open ≤ **?Chosen-Width** is an element of the set of failure hypotheses due to a possible error in **?Min-Open**. By the distance heuristic, it is also the most likely to have failed with **?Min-Open** and **?Chosen-Width** having almost the same value at the time of failure.

The corresponding tuning hypothesis suggests that the tunable parameter **?Chosen-Width** be increased thus making the inequality more likely to hold. A positive slope is then given to the utility function for opening width. Figure 4.13 illustrates the shape of the opening-width parameter's utility function before (top) and after (bottom). The new utility function will now always prefer to choose the widest opening width possible for the grasp. When the refined permissive plan is applied, the resulting gripper finger positions are as illustrated in Figure 4.14.

4.2.1.2. Example 2

Next, after the system has learned to open widely when grasping objects, a second object is presented for grasping. Figure 4.15 shows the system status display while initially planning the grasp for this

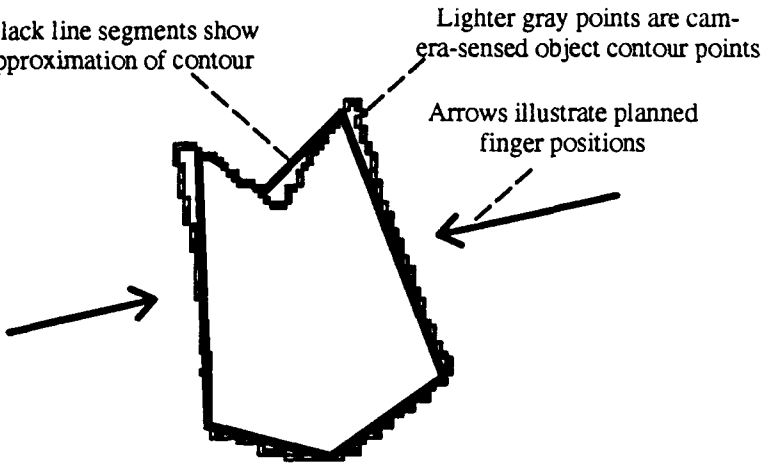


Figure 4.14. A Successful Wide Grasp

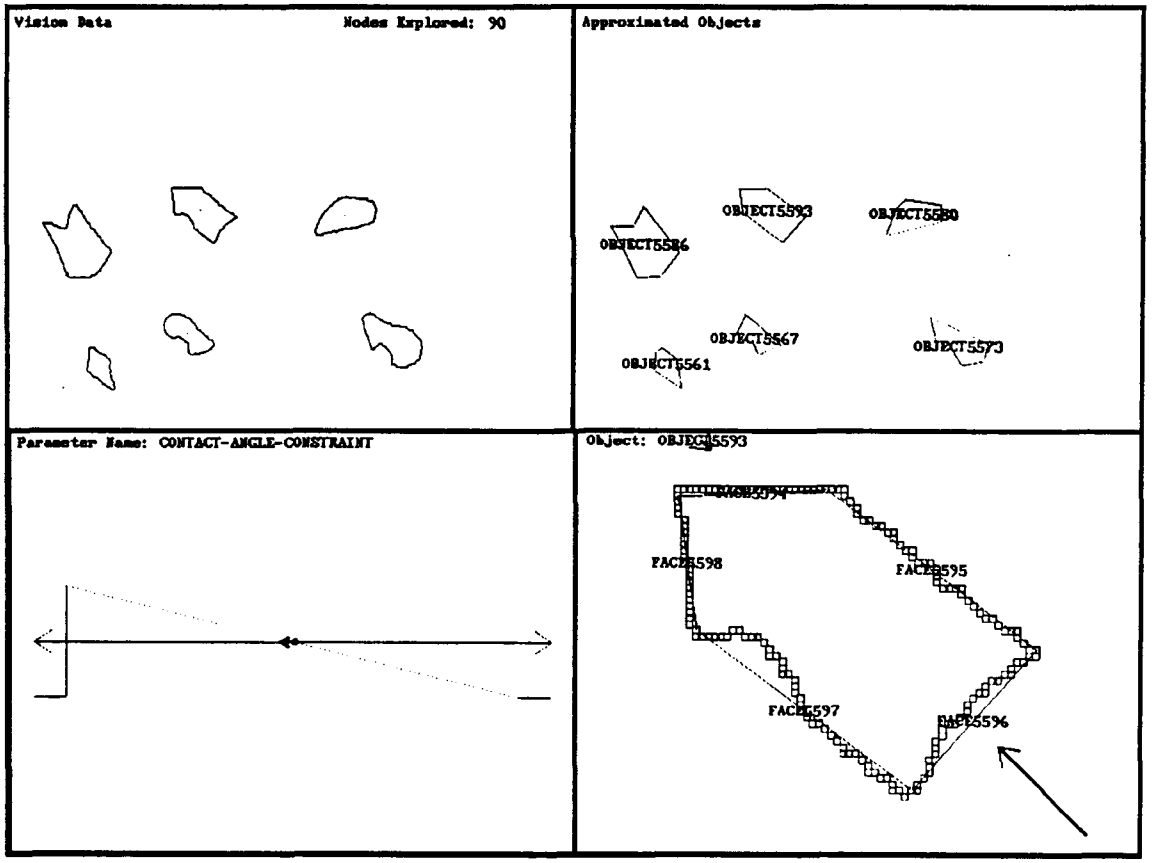


Figure 4.15. System Status Display During Grasp of Object5593

object. Figure 4.16 highlights the selected target object. The dark line indicates the polygonal object approximation and the light colored pixels show the sensed object contour points. The arrows illus-

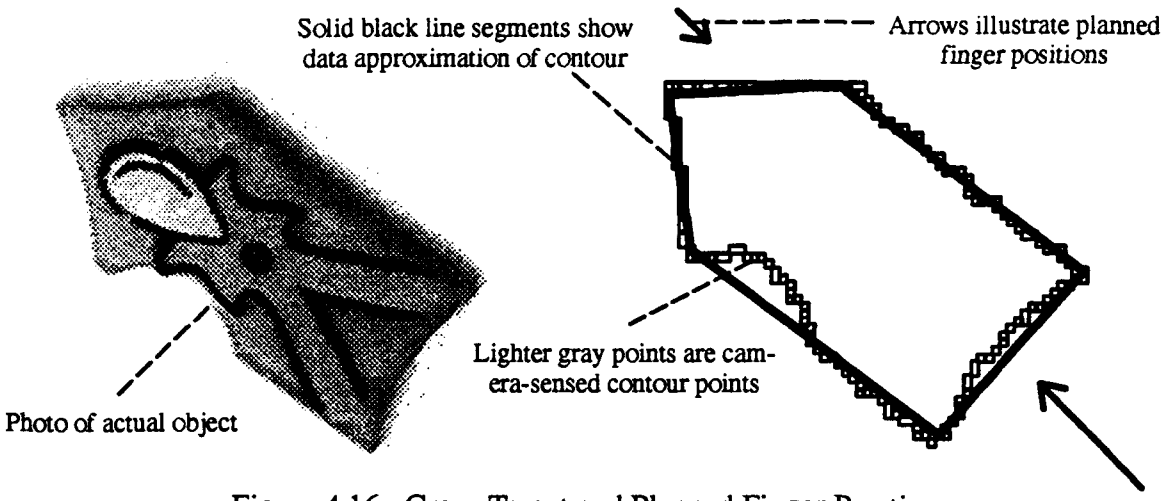


Figure 4.16. Grasp Target and Planned Finger Positions

trate the planned positions for the fingers in the generated grasping plan. Notice, that the fingers are open well clear of the object due to the tuned opening-width parameter learned in the first example. Generally, the opening-width parameter is the first one to be tuned because striking the objects while attempting to surround them is a fairly common error. However, the parameter regarding acceptable angles between contact faces still has only the initial flat utility function. The angle between chosen faces must be greater than 0 (parallel) and less than the angle at which slipping occurs according to the friction coefficient between the gripper and faces being grasped (45 degrees, here). All angles fitting these criteria are treated as equally desirable. Consequently, the two faces chosen for this grasp should be acceptable given the specified friction coefficient. After the explanation is generated, and its associated operator sequence executed, the monitored action shown in Figure 4.17 encounters a violation of the expected sensor readings. The traces in Figure 4.18 and Figure 4.19 show, respectively, plots of gripper force and gripper position with time (in seconds) into the overall execution sequence.

The original explanation for the *stable-grasp* goal indicated in the above monitored action is now suspect due to the violated expectations. A sketch of the specific explanation is shown in Figure 4.20. This explanation for why a stable grasp should have been achieved is the starting point for developing the tuning hypotheses. Explicitly approximate quantities and tunable parameters

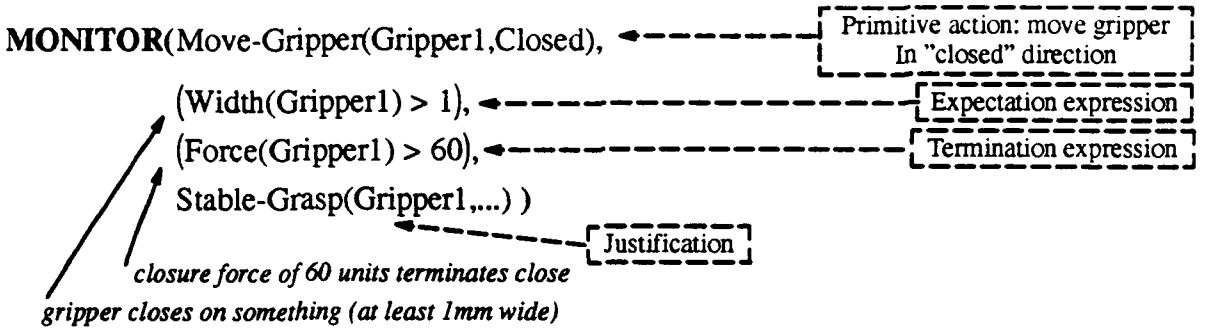


Figure 4.17. The Failing Monitored Action

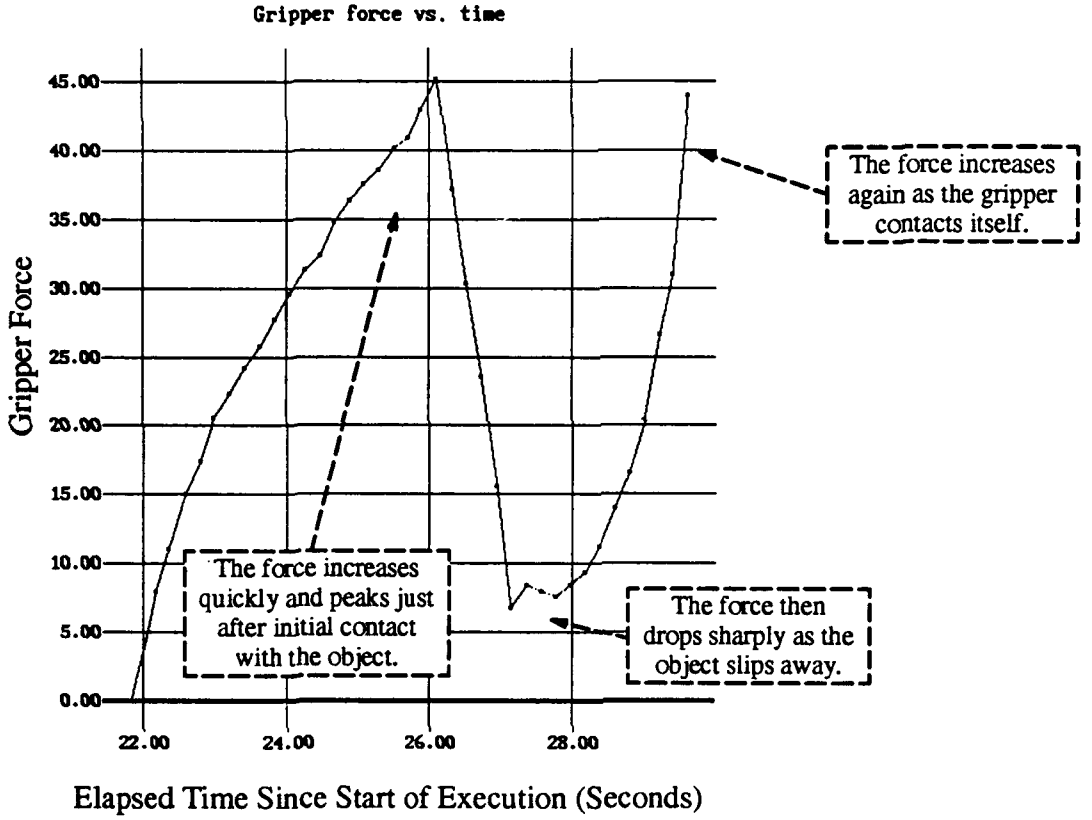


Figure 4.18. Gripper Force vs. Time into Action Sequence

employed in the plan support proof are identified. The angle between the faces selected for grasping is supported by the inequality **?Face-Angle** ≤ **?Friction-Angle** (shown in fully instantiated form at the bottom of the support structure for Stable-Grasp in Figure 4.20). This constraint is one of the set of *Multiple-Leaf-Supports* as described in Chapter 3. The quantity **?Friction-Angle** is computed as the arctangent of the friction coefficient which is a known explicitly approximate quantity. Since

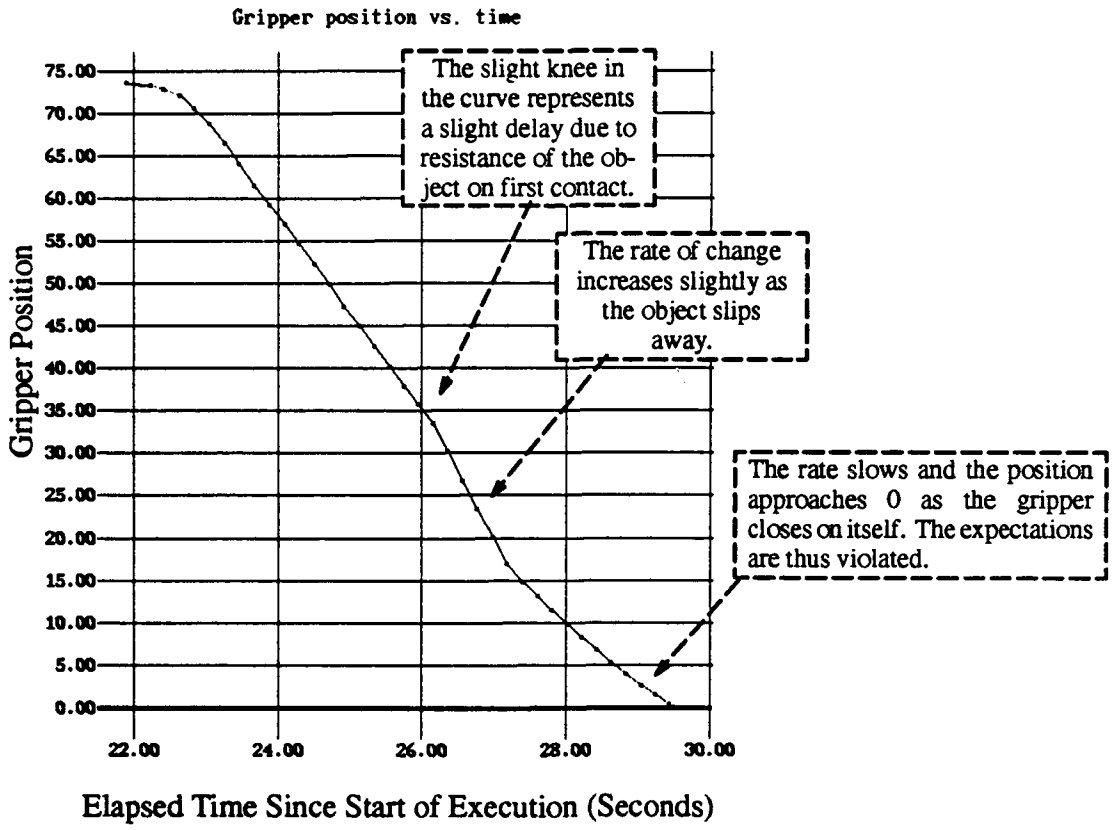


Figure 4.19. Gripper Position (Width) vs. Time into Action Sequence

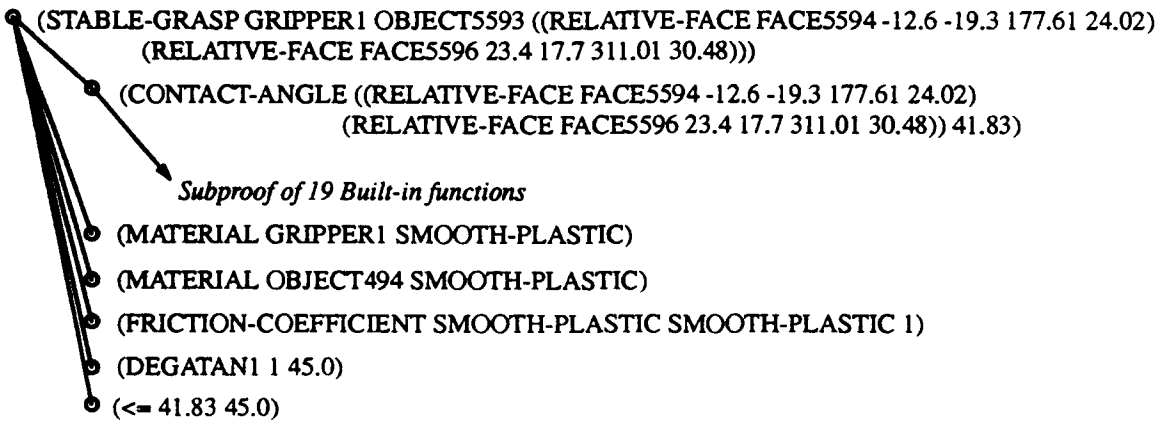


Figure 4.20. Explanation Specific to Failure

there are multiple faces on the object which qualify as having the correct angle, the quantity **?Face-Angle** is a tunable parameter. The constraint **?Face-Angle** ≤ **?Friction-Angle** is a member of the set of failure hypotheses. The specific instantiation of the constraint at the time of failure was

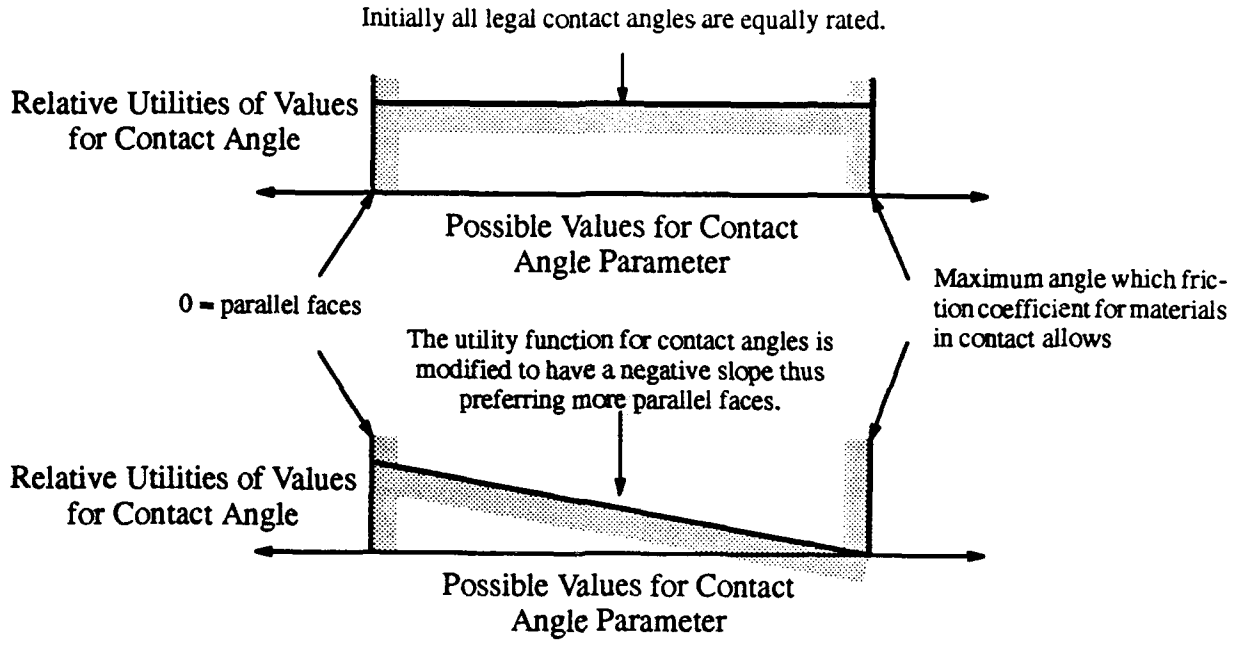


Figure 4.21. The Contact-Angle-Constraint Parameter Utility Function Before and After Tuning

41.83 ≤ 45.0. The values are sufficiently close to make it the leading candidate failure hypothesis. The associated tuning hypothesis suggests decreasing the quantity **?Face-Angle** to make the constraint more likely to be satisfied. This amounts to preferring more parallel sides for grasping. Figure 4.21 illustrates the shape of the contact-angle parameter's utility function before (top) and after (bottom) tuning has occurred. The utility function now prefers the most parallel faces. Figure 4.22

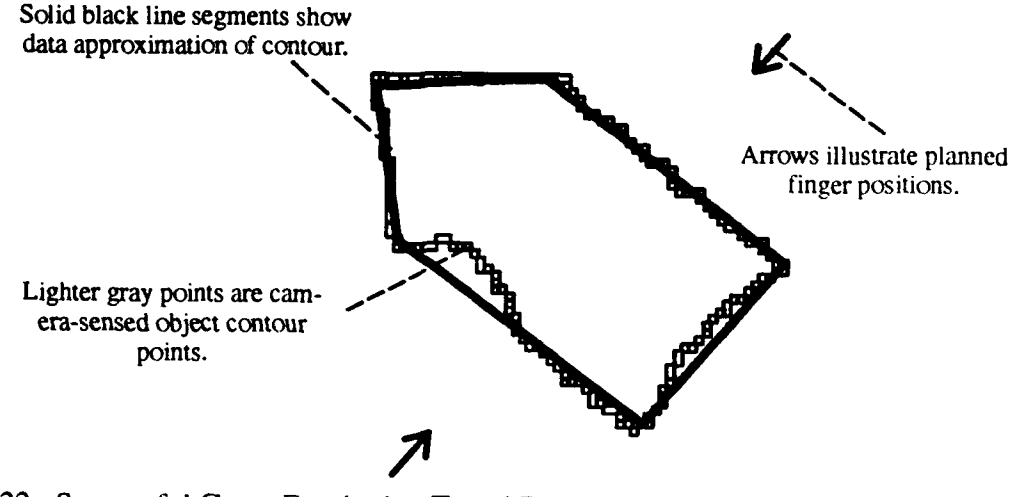


Figure 4.22. Successful Grasp Employing Tuned Parameter Constraining Contact Angle

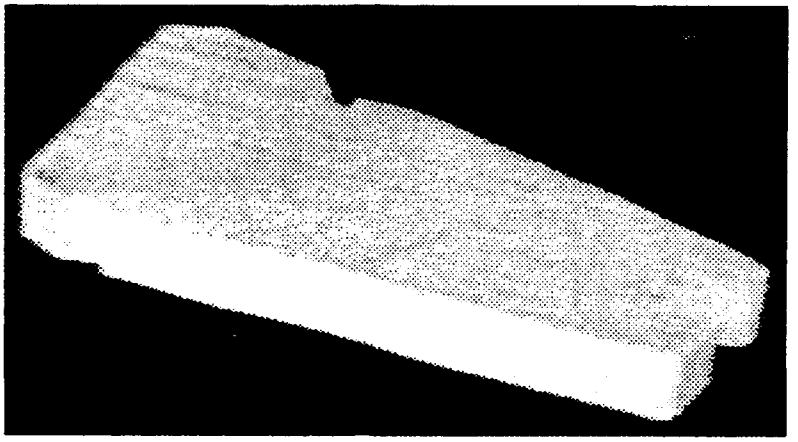


Figure 4.23. A Challenging Piece for the RTX to Grasp

shows the new more permissive plan as applied to the same object. In this case, the most parallel faces are preferred over those picked in the earlier plan application.

4.2.2. Experiment 2

In our second experiment, our goal was to explore tradeoffs between different plan parameters. We, therefore, utilized a set of heavier and larger wooden pieces for which tradeoffs occur. A domain theory was employed that supports legal grasps anywhere along two legal grasping faces of the object. This allows tradeoffs to arise between clearance around the object, the distance from the center of mass of the object, and the force used to squeeze and lift the piece.

4.2.2.1. Example 1

Figure 4.23 shows a large wooden piece which presents a challenging grasp target for our system. It is large and heavy enough that a successful grasp will come very close to the gripper opening and gripper force limitations of the RTX manipulator we employ. It is also rather difficult for a person to predict the best grasping site for the RTX from observation without gathering experience with the object and manipulator. Figure 4.24 shows the target object. The dark line indicates the polygonal object approximation and the light colored pixels show the sensed object contour points. The arrows

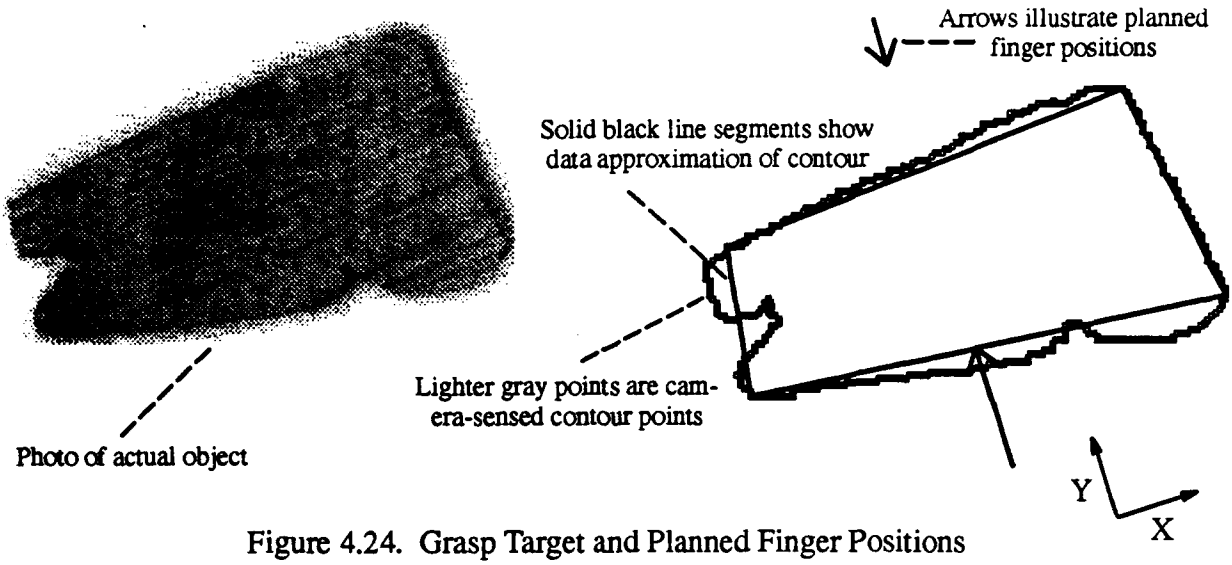
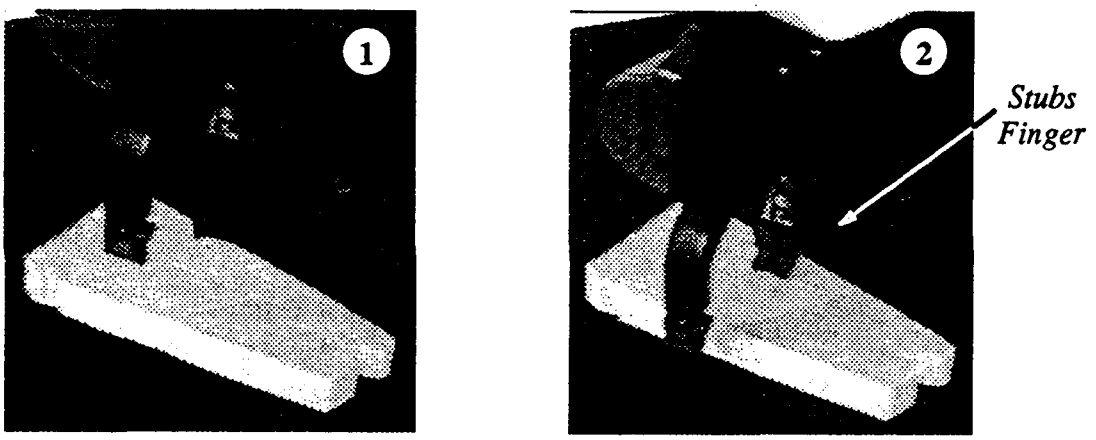


Figure 4.24. Grasp Target and Planned Finger Positions

illustrate the planned positions for the fingers in the generated grasping plan. The domain theory we employed for oversized wooden pieces had added flexibility in allowing the grasp center to be chosen anywhere along the chosen sides. This results in four parameters for this example: X (along the major axis of the piece), Y (along the minor axis), Clearance (the minimum distance between any gripper finger and the object while surrounding the piece), and Force (the duty-cycle given to the motor while closing the gripper in grasping the piece). In the case of this object, only the two long sides of the four-sided object approximation afford potential grasping positions. The pair of short sides is too far apart for the gripper to surround. Any pairing of long and short sides results in a contact angle too large for a stable grasp. Consequently, there is only one viable choice of faces and face angle, unlike the examples of Experiment 1. The planned grasp of Figure 4.24 is near the center of the piece but is centered toward the top. The gripper is open the maximum amount and a minimum amount of force will be applied to lift the object. It is important to realize that the system starts with no preferred values for the parameters. The initial arbitrary ordering of the constraints as passed to the SIMPLEX algorithm in the implementation results in this choice of initial grasp parameters. Figure 4.25 shows execution of the plan. While attempting to surround the piece, the expectations are violated when contact is made prior to reaching the table. As this example involves



Before Surrounding Piece *While Attempting to Surround Piece*

Figure 4.25. Execution of the First Planned Grasp

several steps, we will omit the force and position plots which show the expectation violations. The plots and expectations are similar to those seen in the previous experiment.

A number of competing failure and tuning hypotheses are generated. The one the system rates as the most likely to eliminate the failure increases the clearance between the gripper and object. The quantity Clearance is a parameter of the plan. A preference is installed to prefer the maximum clearance for this plan.

The system once again uses the camera to look at the workspace since the object may have been moved by the last failed grasp. The target object is once again approximated from the camera data. The permissive plan is again applied to grasping the object. Figure 4.26 shows the resulting planned grasp. In this grasp, the narrowest end of the object is preferred as it provides the desired maximum clearance and no preference has yet been expressed for the other plan parameters. In this case, the X parameter is at a minimum, the Y parameter is in the middle of its range, and the force applied is at a maximum. The minimum force required is a function of the distance from the center of mass. A large force is required here because the planned grasp is far from the center of mass of the piece. The photo sequence of Figure 4.27 shows execution of the planned grasp. Unlike the first planned

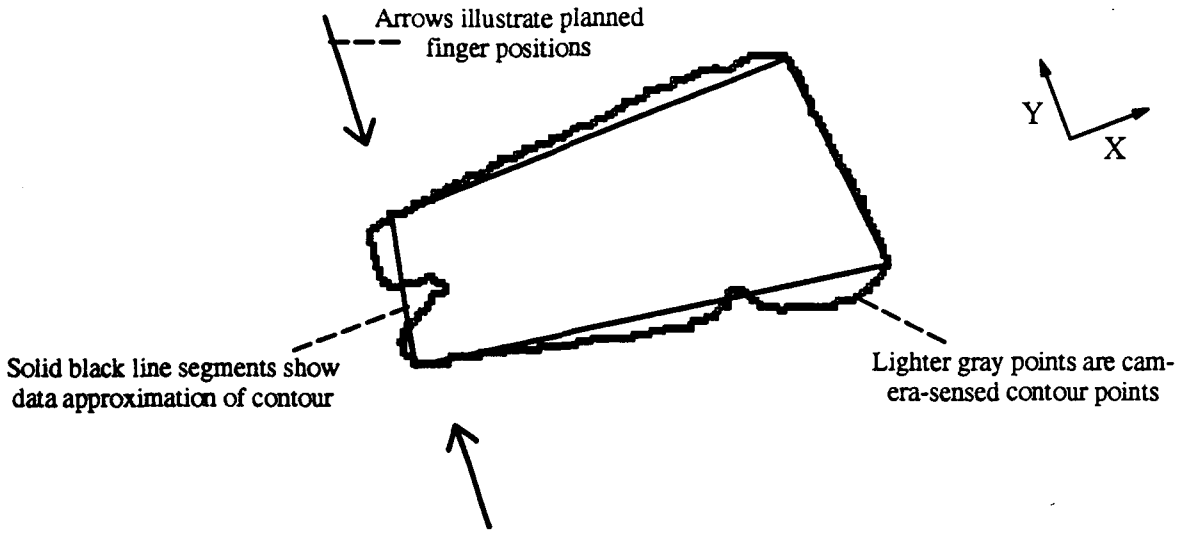


Figure 4.26. Grasp Target and Second Planned Finger Positions

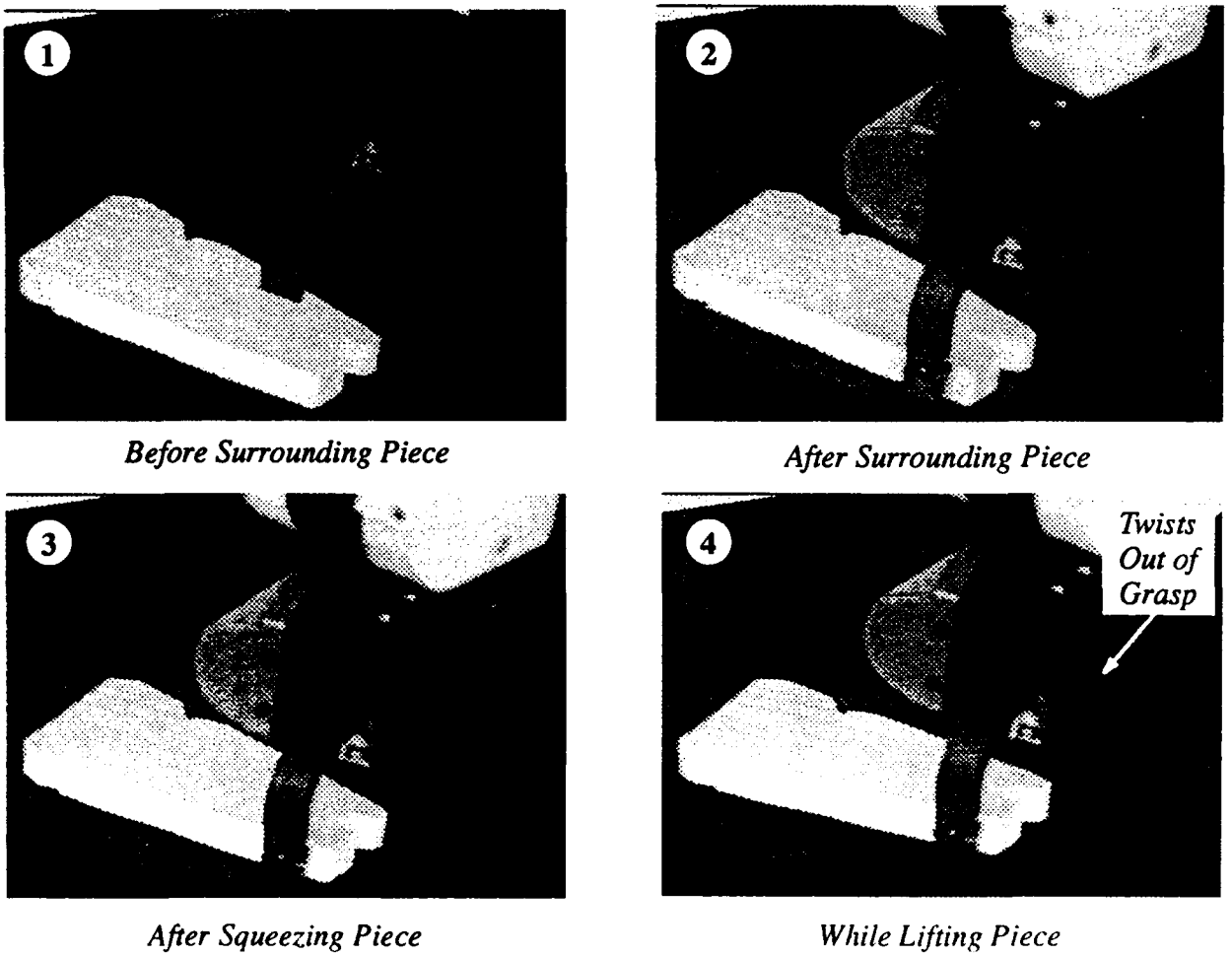


Figure 4.27. Execution of the Second Planned Grasp

grasp, the gripper is able to successfully surround the piece and establish contact with it. However, when it comes to lifting the piece, the gripper does not impart sufficient force to counteract the torque of the piece and the piece twists out of grasp (and falls from the gripper as the gripper is lifted higher). The plan has an expectation that as the piece is lifted contact is maintained. In part, this is justified by a proof that the forces and torques balance so that the piece may be lifted and remain stable. In this case, this expectation is violated. A number of competing failure and tuning hypotheses result from an analysis of the plan. The tuning hypothesis chosen as most likely to remedy the failure is the one which prefers larger X values. A better tuning hypothesis would have been to increase the amount of force applied by the gripper. However, the force was already at a maximum and could be increased no further. A preference for large X values will ensure that the selected grasp lies closer to the center of mass of the object than this failed grasp. The preference for increasing X values is added to the permissive plan.

The system once again applies the plan for grasping the object. This time the planned grasp position is as illustrated in Figure 4.28. The chosen grasp attempts to maximize the values for the Clearance and X plan parameters. The Y value is in the middle of the range (to maximize clearance) and the force is significant but not at a maximum because the grasp is closer to the center of mass. The se-

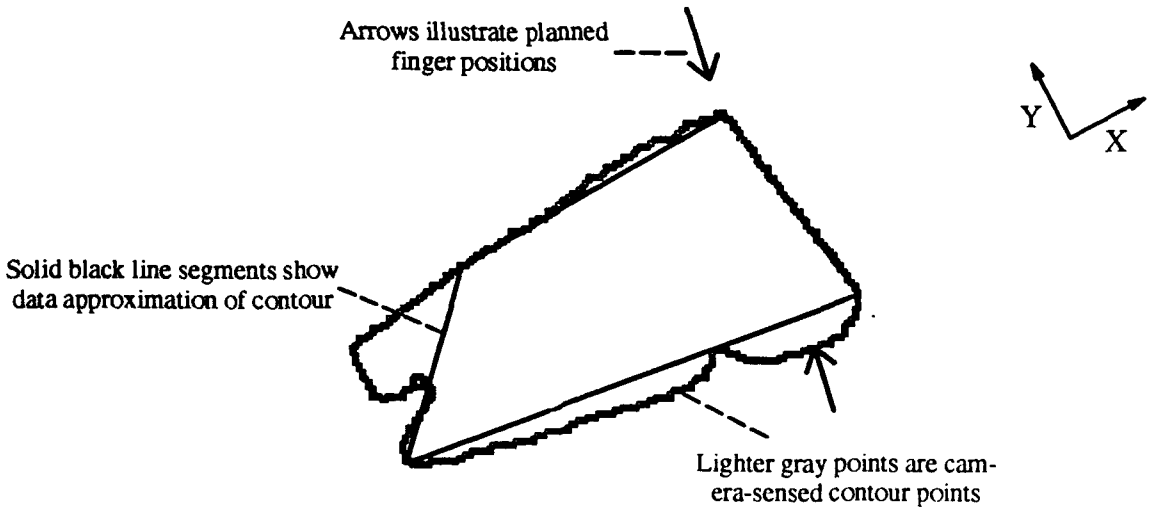
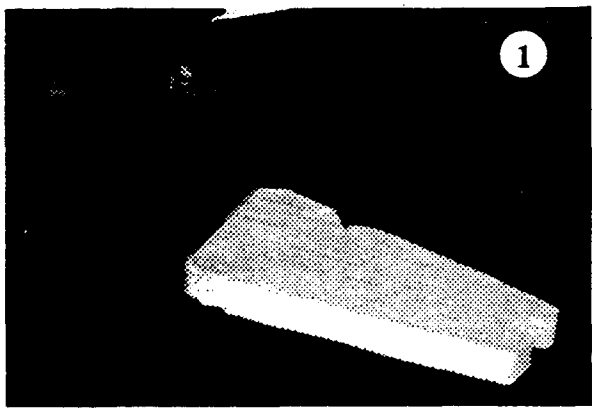
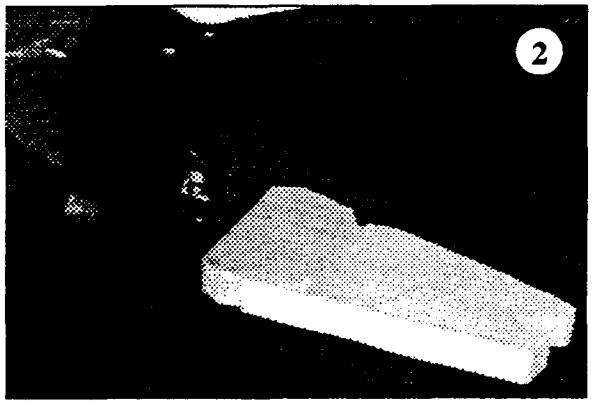


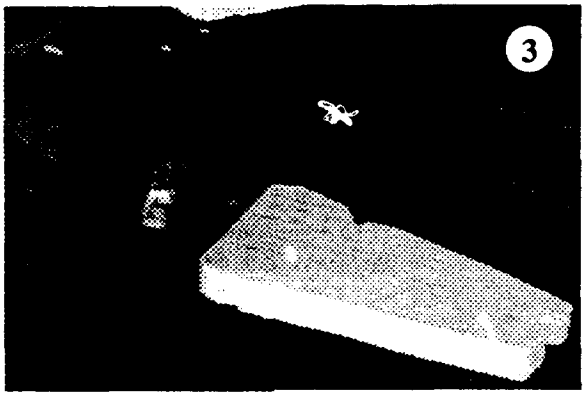
Figure 4.28. Grasp Target and Third Planned Finger Positions



Before Attempting to Surround Piece



After Attempting to Surround Piece



After Attempting to Make Contact With Piece

Figure 4.29. Execution of the Third Planned Grasp

quence of photos shown in Figure 4.29 shows execution of the plan. In this case, the gripper misses the piece entirely, surrounding empty space and closing. The expectation is that contact be established with the piece while closing. This expectation is based on a proof that the space occupied by the gripper fingers and the approximation to the piece must intersect when the gripper closes after surrounding the piece. This expectatio· is violated. The leading tuning hypothesis is to decrease the value of the X parameter. This new preference combined with the previous preference for large X values gives a two-piece linear preference function with a peak at the central possible X value.

The permissive planner is again applied to the object and results in the grasping position illustrated in Figure 4.30. Here, a central grasping position is chosen with a large clearance. A minimum force is used since the position is close to the center of mass. The photo sequence of Figure 4.31 depicts

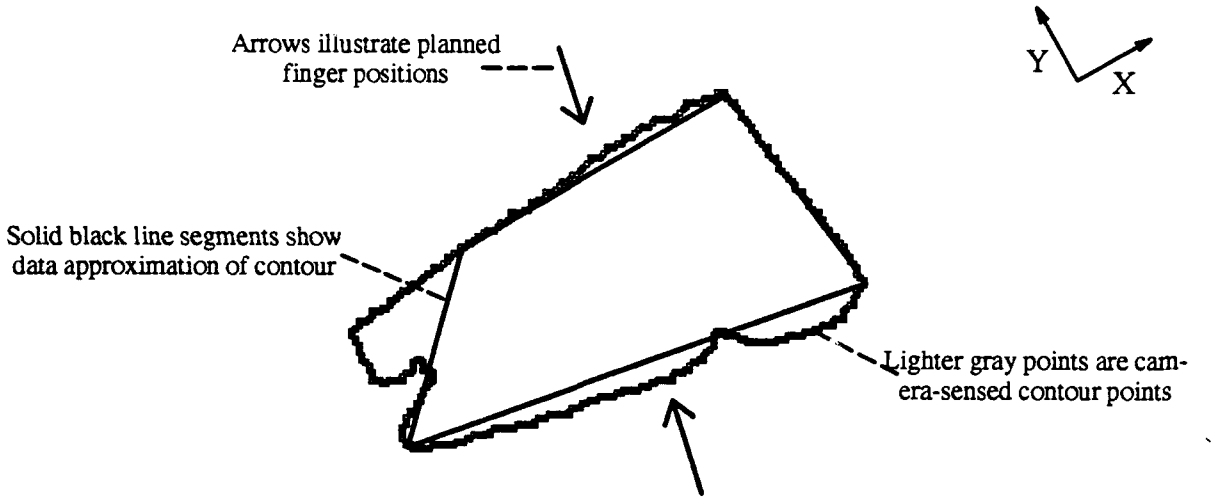
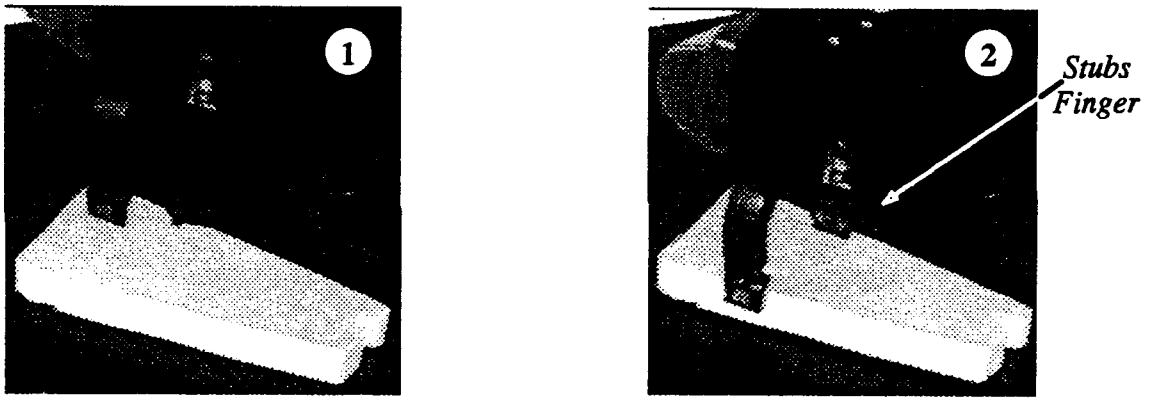


Figure 4.30. Grasp Target and Fourth Planned Finger Positions



Before Attempting to Surround the Piece *While Attempting to Surround the Piece*

Figure 4.31. Execution of the Fourth Planned Grasp

the result of this grasping attempt. The gripper stubs one finger on the piece in attempting to surround the object. This violates the expectation that no collision occur while surrounding the piece. The leading tuning hypothesis indicates the value of the X parameter should be decreased. This will allow further clearance around the piece. At the same time, an earlier preference exists for increasing the X parameter value due to the failure at the small end of the piece. The two-piece linear preference function for the X value is adjusted to have its peak halfway between the point which just failed and the narrow end of the piece.

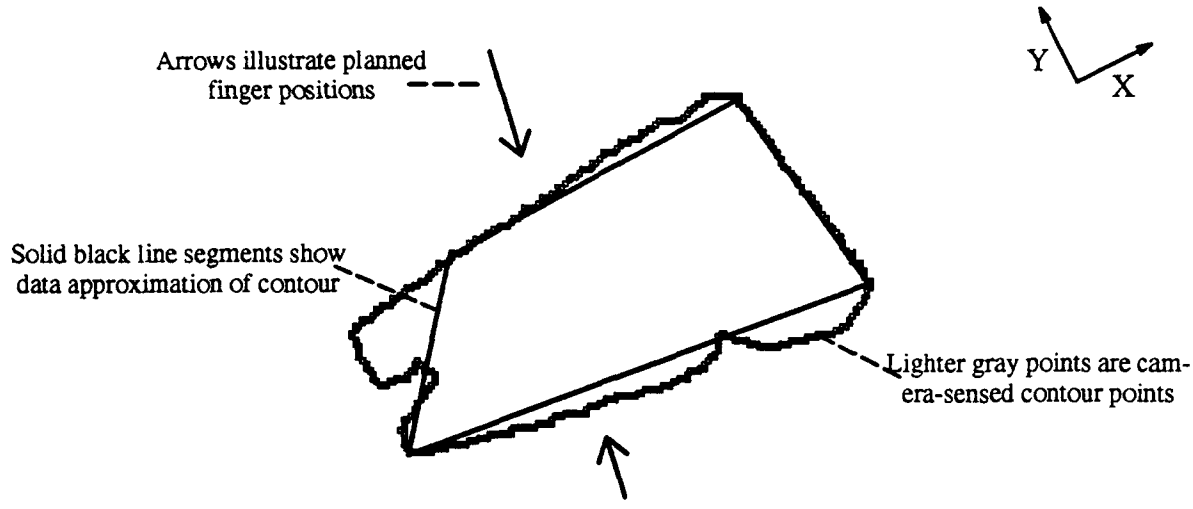
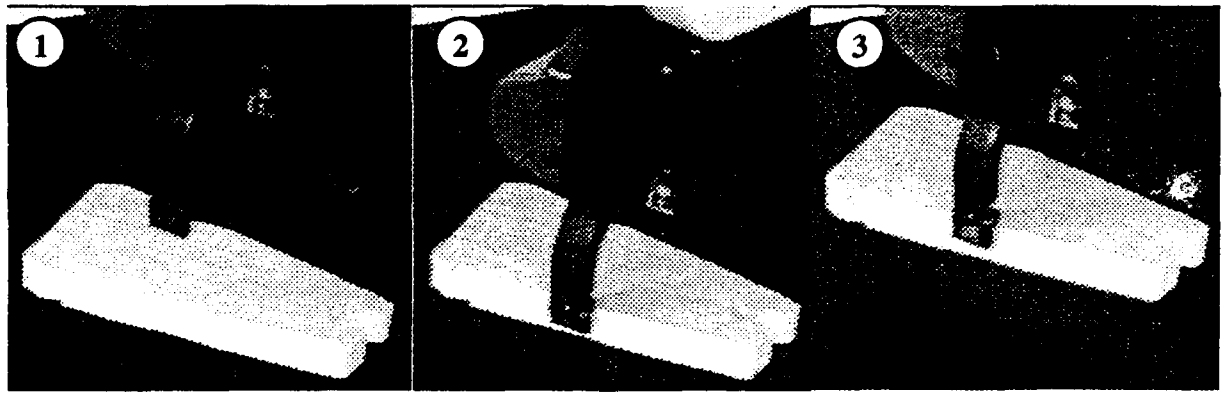


Figure 4.32. Grasp Target and Fifth Planned Finger Positions

The permissive plan is applied again to grasping the object and results in the grasp indicated in Figure 4.32. This grasp has the gripper open to the maximum, is closer to the narrow end promoting a larger clearance around the piece, and is closer to the center of mass giving a higher likelihood of a stable grasp. The best point to grasp the object is dependent on a number of quantities for which only approximate values are known including values not easily given by inspection such as the friction coefficient and center of mass. The photo sequence of Figure 4.33 shows that the permissive planner has finally arrived at a successful grasping plan for this object.



Before Surrounding Piece *After Surrounding Piece* *After Lifting Piece*

Figure 4.33. Execution of the Fifth Planned Grasp

4.2.2.2. Example 2

We presented a similar object to the system. The same permissive plan applied and gave the grasp positions indicated in Figure 4.34. The photo sequence of Figure 4.35 shows the successful resulting grasp.

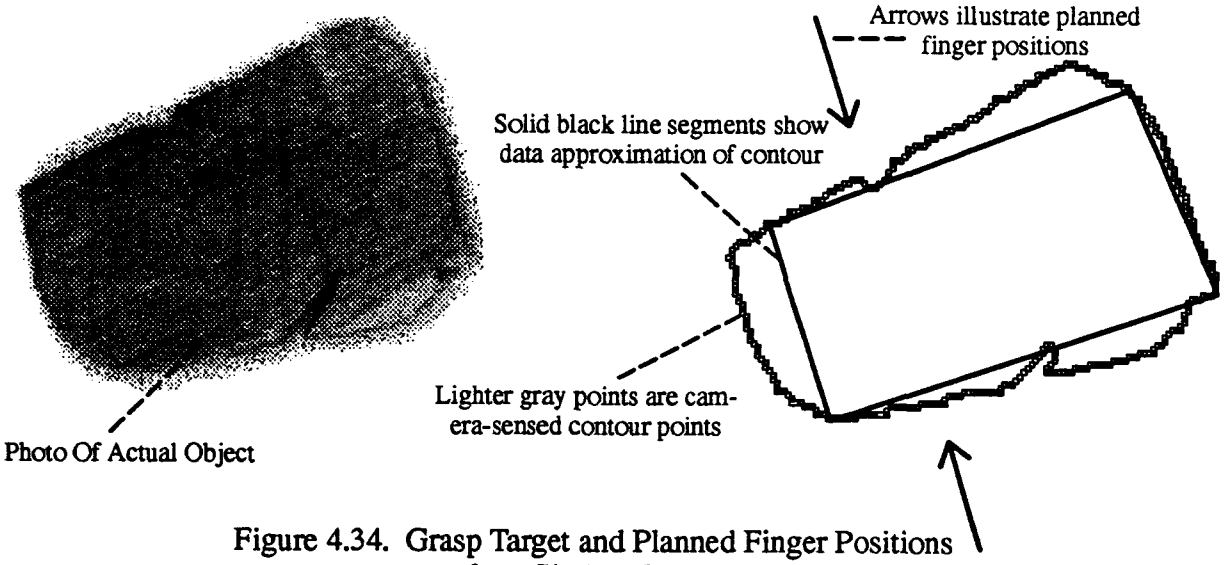
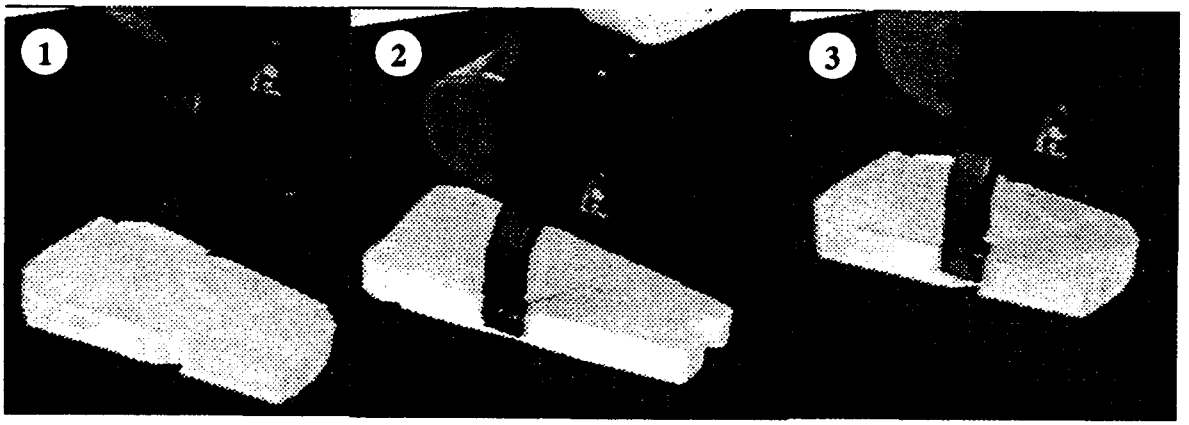


Figure 4.34. Grasp Target and Planned Finger Positions for a Similar Object



Before Surrounding Piece *After Surrounding Piece* *After Lifting Piece*

Figure 4.35. Execution of the Grasp for a Similar Object

4.3. Empirical Testing

The GRASPER system was given the task of achieving equilibrium grasps on the 12 smooth plastic pieces of a children's puzzle. Figure 4.36 shows the gripper and several of the pieces employed in

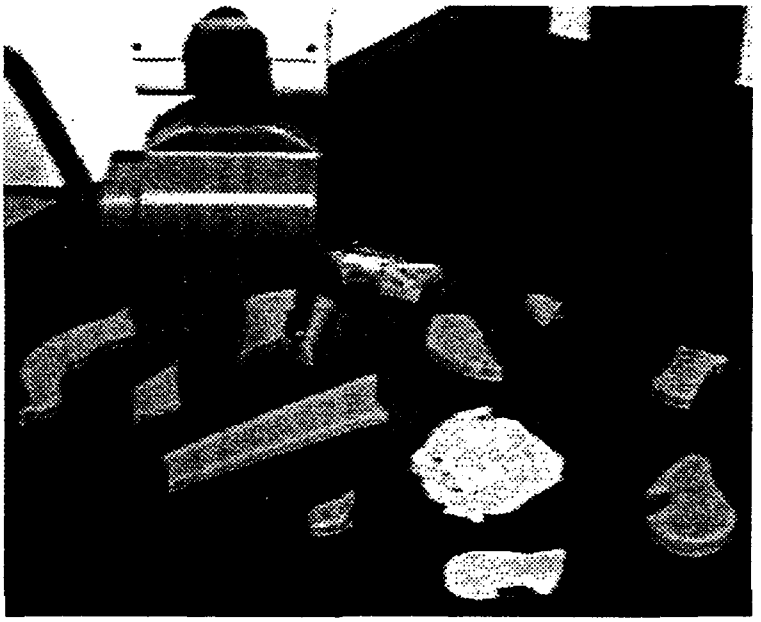


Figure 4.36. Gripper and Pieces

these experiments. A random ordering and set of orientations were selected for presentation of the pieces. Target pieces were also placed in isolation from other objects. That is, the workspace never had pieces near enough to the grasp target to impinge on the decision made for grasping the target. The first run was performed with plan refinement turned off. The results are illustrated in Figure 4.37. Failures observed during this run included *finger stubbing failures* where a gripper finger struck the top of the object while moving down to surround it. Such a failure is depicted in the sequence of photos in Figure 4.38. Also observed were *lateral slipping failures* where, as the grippers were closed, the object slipped out of grasp, sliding along the table surface. Figure 4.39 shows one such failure. In the system's initial approximate representation for the world, the choice of grasping faces is constrained only by the gripper being able to open wide enough to surround them and that an equilibrium grasp is realizable with the current gripper-object friction coefficient (initially 1

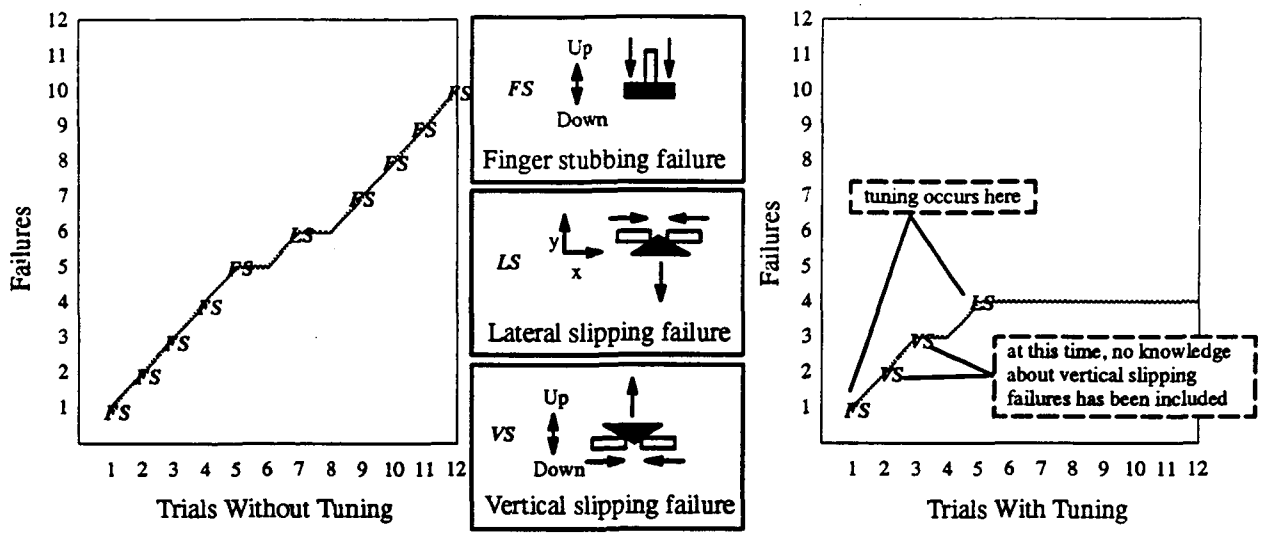


Figure 4.37. Comparison of Tuning to Nontuning in Grasping the Pieces of a Puzzle

Figure 4.38. An Instance of a Finger Stubbing Failure

Figure 4.39. An Instance of a Horizontal Slipping Failure

here). Since a friction coefficient of 1 is likely to be high for these materials, the choice of contact faces is likely to be under-constrained initially, resulting in slipping failures. The choice of opening width is the minimum deviation from the current opening width (initially 0 here) which satisfies the approximate model of the grasp. Due to uncertainties in the world, this opening width may often result in stubbing failures. Therefore, the error rate of our initial approximate plan was high resulting in nine finger stubbing failures and one lateral slipping failure in 12 trials.

In our second run, refinement was turned on. An initial stubbing failure on Trial 1 led to a tuning of the chosen-opening-width parameter which determines how far to open the gripper for the selected grasping faces. Since the generated tuning hypothesis indicated that opening wider would decrease the chance of this type of a failure, the system tuned the parameter to choose the largest opening width possible (constrained only by the maximum gripper opening and possible collisions with nearby objects). In the case of isolated grasp targets, opening to the maximum gripper width is preferred. In Trials 2 and 3, finger stubbing failures did not occur as they had previously because the opening width was greater than the object width for that orientation. *Vertical slipping failures*, which the current implementation does not currently have knowledge about, did occur. Such a failure is illustrated in Figure 4.40. The system had to be told that a vertical slipping failure had occurred instead of the lateral slipping failure it thought had occurred, because, without further knowledge about vertical slipping failures and a means for detecting them, they look in other ways (the force vs. position profile of the gripper closing) like a lateral slipping failure. Preventing vertical slipping failures involves knowing shape information along the height dimension of the object. This could be accomplished using a 3D sensing device like a laser scanner in a possible future extension of the system. In Trial 5, a lateral slipping failure is seen and the tuning hypothesis is to decrease the contact angle between selected grasping surfaces through tuning the contact angle parameter. This is tuned to prefer smaller contact angles. A single tuning for the finger stubbing and lateral slipping failures was sufficient to eliminate those failures with isolated grasp targets.

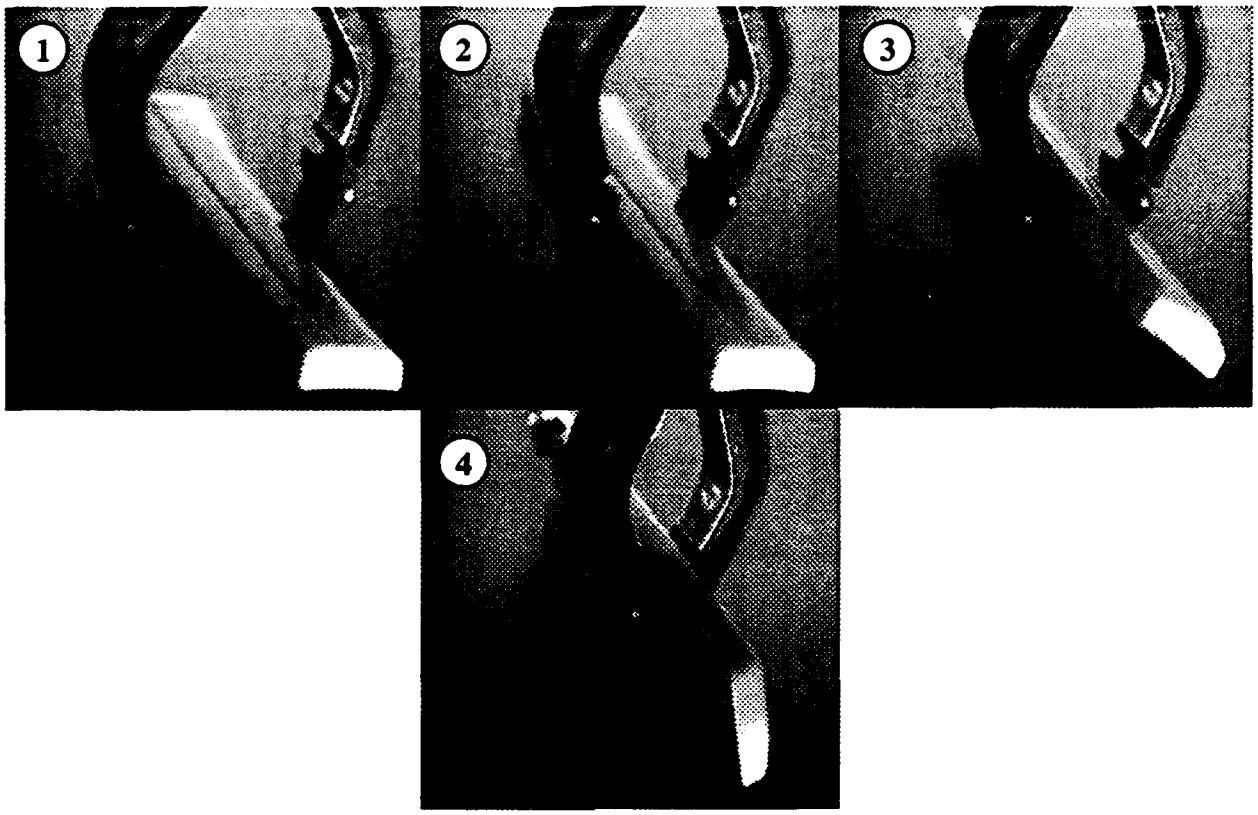


Figure 4.40. An Instance of a Vertical Slipping Failure

The permissive planner performed well in the grasping domain and was able to quickly improve plans by identifying and implementing tuning hypotheses. These hypotheses were identified using the expectation failure and the supporting explanation for the plan. In the next chapter, where we explore part orientation using a tiltable tray, we will have the opportunity to compare our approach to one which abandons a classical domain theory in favor of tabulation of probabilities. This will further highlight the significance of the permissive planning approach which can maintain classical projection while refining plans for improved real-world performance.

5. PERMISSIVE PLANNING IN THE TRAY-TILTING DOMAIN

In automated manufacturing, it is often necessary to take randomly oriented parts and deliver them in a specific orientation. One might use a vibratory feeder to accomplish this but the feeder design would have to be customized for every type of part which might be employed. More general techniques can be employed where a robotic manipulator can be used to orient parts without requiring physical changes to the line to accommodate different parts. Generally, this requires a skilled programmer to construct a program for orienting a specific part and to make the program robust enough to handle likely contingencies. Manipulator part orientation can use fences which when swept against a side of the part can cause it to come into a predictable alignment. Plans can then be developed taking into account the forces involved to theoretically achieve the desired orientation (in the model). However, the real operating environment is inherently complex. The models used to construct plans then tend either to be too simplistic to yield good results in the real world or they tend to be very expensive taking into account numerical uncertainty ranges on all of the sensed data, modelling the possible vibrations of a conveyer moving the part, etc. This has led to an interest in systems which can be trained to perform tasks such as these and to improve their performance through experience.

Tray tilting has been investigated both analytically and empirically by a number of researchers [Christiansen, Christiansen90, Erdmann, Taylor87]. Following the approach of Christiansen, we constructed a manipulator-controlled tiltable tray to allow a direct comparison of the permissive planner with the stochastic approaches that use such a setup. Our setup is illustrated in Figure 5.1 and consists of a SCARA-type robotic manipulator controlling an 11x11 inch aluminum tray. Our experiments were performed with a $1\frac{1}{8}$ x $1\frac{1}{8}$ x $2\frac{5}{16}$ inch wooden block. An overhead camera and vision system is used to sense the part configurations. In the tiltable tray, the tray's sides serve the same function as manipulator controlled fences do for part orientation. The tray-tilting setup

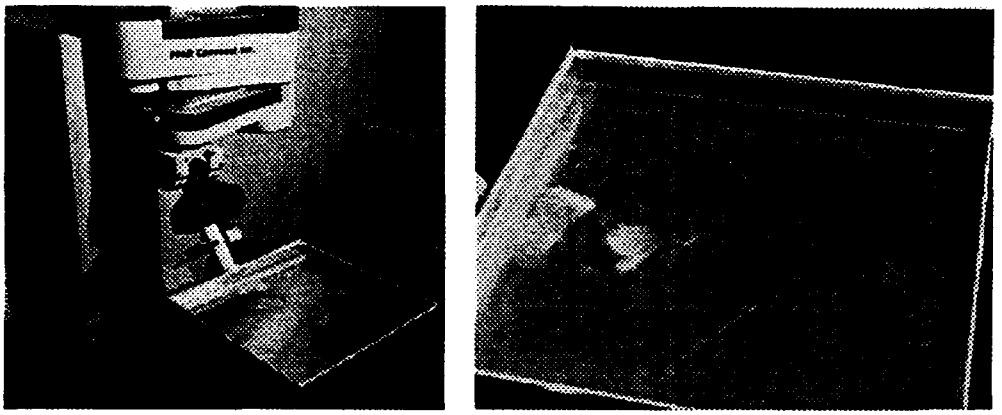


Figure 5.1. Setup for Tray-Tilting Domain

represents a fairly complex environment. The manipulator controlling the tray is subject to control errors which affect tray orientation[13], vision system sensor errors (shadows, poor lighting, etc...), and a simplistic model of the rather complex aluminum tray (which is not uniformly smooth) and of the piece being manipulated. Figure 5.2 shows a tilting sequence where the block spins in a hard-to-predict way due to surface imperfections. This setup provides a good test for any learning system seeking to function in a complex, uncertain environment.

As we are comparing a probability-based stochastic approach to permissive planning in this domain, we will first introduce the stochastic approach. Next, we discuss applying permissive planning in this domain. This is followed by a detailed example contrasting the two approaches. Lastly, we present the results of large numbers of empirical trials on both techniques.

5.1. The Stochastic Approach

In complex domains with uncertainty, the same action performed twice, from what is recognized as the same state of the world, may lead to different states. Stochastic approaches seek to model the likelihoods of arriving at different states given a starting state and action. Data can be gathered

13. Control is different in our setup as compared with the CMU tray-tilting setup [Christiansen90] because our tray is handled by the manipulator like a frying pan and needs to simultaneously move a number of joints to achieve the desired tilt.

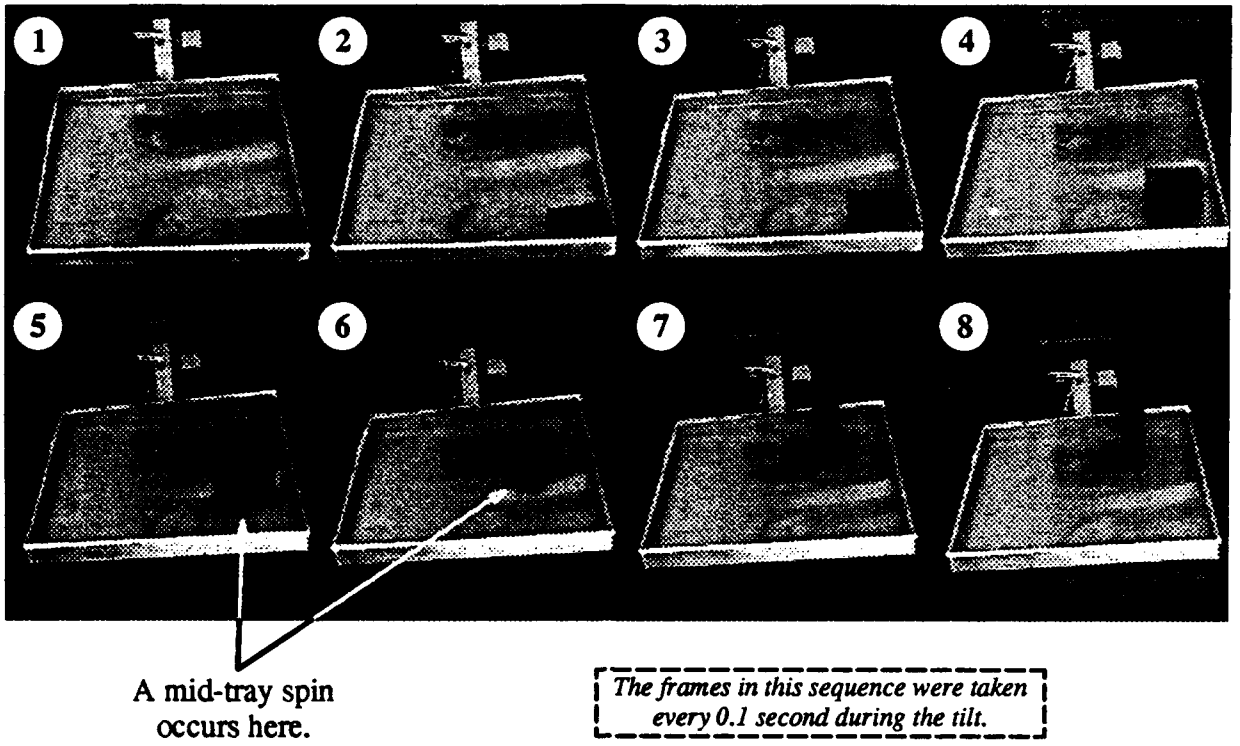


Figure 5.2. A Tray Imperfection Imparts a Difficult-to-Predict Spin

through experience with a task which can enable state transition probabilities to be calculated. These probabilities can be employed by a stochastic planner to produce plans which seek to accomplish goals in uncertain environments. Probabilities are a powerful tool to summarize complex behaviors in the domain which might be expensive to model explicitly. Stochastic techniques are an attempt to move away from complex domain theories which render classical planning intractable. Unfortunately, the stochastic techniques have problems as well. Any learning which occurs has an implicit context. Applicability conditions for a particular plan are not learned. Gathering state transition probabilities requires large numbers of trials. Due to the complexities of the algorithms used in planning with transition probabilities, discrete world state representations must be employed, often at a very coarse level.

The stochastic planning algorithm we use for comparison is the one used by Christiansen and Goldberg [Christiansen90]. It finds an optimal n-step plan for accomplishing a goal given a complete

state transition matrix. Let P_a be the stochastic transition matrix where p_{ij} is the conditional probability that state j is achieved from state i with action a. This representation assumes a finite set of world states and actions. In the tray-tilting domain, the world is discretized into 18 possible states as shown in Figure 5.3. There are horizontal and vertical configurations for the rectangular block in each of nine sectors. As with Christiansen's setup, the middle states and states with the block perpendicular to the center of a side are difficult to achieve. We therefore also exclude them from our experiments. We also choose the same set of discrete actions as Christiansen: 12 tilt azimuths spaced every 30 degrees.

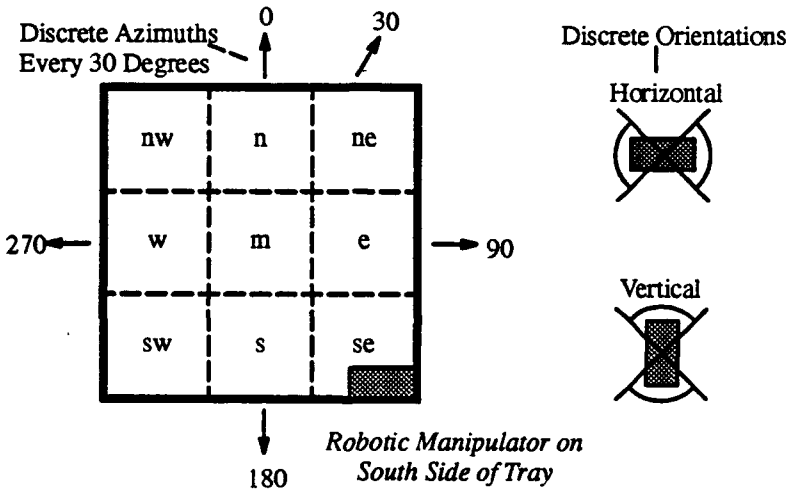


Figure 5.3. Discretization of Tray States and Actions

Our stochastic transition matrix P_a was computed using 3000 training trials entering the observed probabilities in the matrix. Stochastic transition matrices for each of a sequence of actions can be combined by matrix multiplication to yield transition probabilities across the action sequence. For instance, $P_{a_4,a_7,a_2} = P_{a_4}P_{a_7}P_{a_2}$ gives a transition matrix which represents the state transition associated with the action sequence $< a_4, a_7, a_2 >$. If the system starts out in state i before the action sequence and it is desired to achieve state j, the p_{ij} entry of P_{a_4,a_7,a_2} gives the probability of that transition. Therefore, in seeking the best set of n fewer actions to accomplish a specific transition,

the algorithm Christiansen proposes [Christiansen90] is to generate all possible stochastic transition matrices for action sequences less than or equal to n in length and to observe the probability entry corresponding to the desired transition. The sequence which maximizes this probability is chosen. The complexity of this algorithm is $O(n^2m^k)$ for a k-step plan with n world states and m actions. The algorithm is expensive but produces k-step plans which are optimal given the state transition data. We contrast this with a planner which employs approximations to quickly construct unguaranteed plans. The primary focus of our comparison with the permissive planning technique is on the success rate of the resulting plans.

5.2. The Permissive Planning Approach

Through use of the overhead camera, the block's position and orientation in the tray can be sensed. The permissive planning system was given one of the 12 goal configurations of the stochastic system. It was also given the approximate position and orientation of the block in the tray to the best of the vision system's resolution. The only operator available to the permissive planner is the tilt operator. The steepness of the tilt was fixed in this case to 35 degrees. This was steep enough that the block always slides but not so steep that it falls out of the tray. The direction of the tilt could be any continuous value between 0 and 360 degrees. This contrasts with the stochastic approach which was limited to a discrete set of tilt angles in order to make the approach tractable. The use of a domain theory in permissive planning makes such a discretization of the tilt angle unnecessary. In every situation encountered, there was a range of possible tilt directions which would achieve the result of moving the block into the desired one of 12 goal positions. The tilt direction was therefore a tunable parameter on these problems.

In the grasping domain, action expectations included monitoring sensors on the manipulator during execution of the action. In the tray-tilting domain, expectations consist of confirming that the block ends in a configuration within some set of acceptable configurations after each tilt. Since, one-tilt

plans were being performed by the permissive planner, here this amounts to identifying which of the 12 goal configurations results (as one tilt allows no intermediate resting places for the block). The set of expectations for achieving a certain configuration consists of six inequality constraints which are expected to hold: two to bound the angle which the block is in, and four to bound the position of the center of the block in the tray. Each of these inequalities is justified by some part of the explanation supporting the plan. For instance, suppose the block was starting in the northwest horizontal configuration shown in Figure 5.4 and was desired to be in the southeast vertical configura-

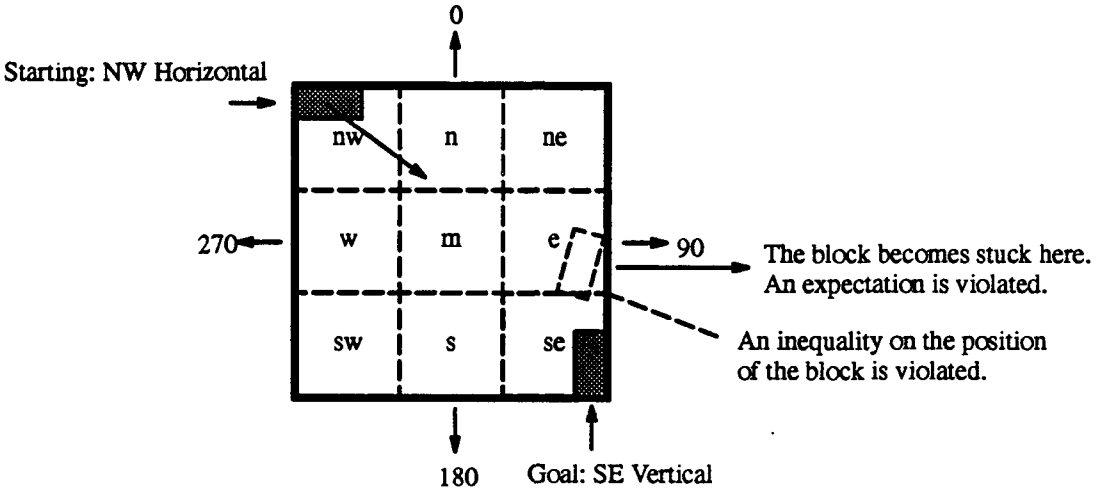


Figure 5.4. An Example of Tilting to Achieve a Goal

tion. The explanation justifying the permissive plan may involve tilting the tray toward 100 degrees so that upon collision with the east wall the block pivots and then slides into the southeast corner while oriented with its long side to the east wall. In this way, it would end in the desired vertical configuration. If, however, the expectation failed and the position of the block was found to be in the east vertical configuration as shown by the dashed outline of the block in the figure, an inequality representing the bound between the east and southeast configurations is violated. This relates to a part of the explanation which justifies why the block should have continued to slide into the southeast corner by overwhelming the frictional forces present with the bottom and side of the tray. One possible failure hypothesis is that the actual angle the tray was tilted (an explicit approximation) did not have a steep enough component toward the south end of the tray. The implementation for

tray-tilting differs in three ways from that described for grasping in Chapter 4: the physical setup of the tray-tilting system is different, a domain theory for tray tilting has been provided, and the expectations take the form of inequalities bounding a set of expected final positions and orientations for the block.

5.3. A Comparative Example

Before presenting comparisons results, let us consider how stochastic and permissive planning compare on an example problem. The problem illustrated in Figure 5.5 involves moving a rectangular

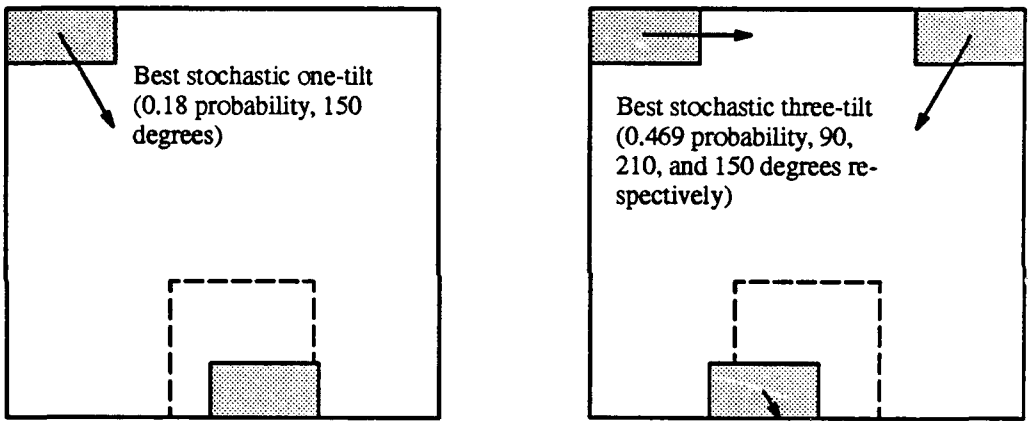


Figure 5.5. Stochastic Tilting Plans for Achieving the South Horizontal Configuration from the Northwest Horizontal Initial State

block from a northwest horizontal configuration to a south horizontal configuration. The stochastic transition matrix indicates the best one-tilt stochastic plan is to tilt the tray at 150 degrees to achieve the goal. The probability of this being achieved is rated at 18%. Since, the complexity of the stochastic algorithm increases significantly as the number of available actions increases, the limitation to 12-tilt direction angles is evident. The three-tilt stochastic plan indicates the sequence of tilts <90,210,150> and has an estimated probability of success of 46.9%. Note that the three-tilt plan establishes that approaching the south sector from the northeast sector has a higher probability of success than a direct approach from the northwest sector. This can be due to characteristics of the

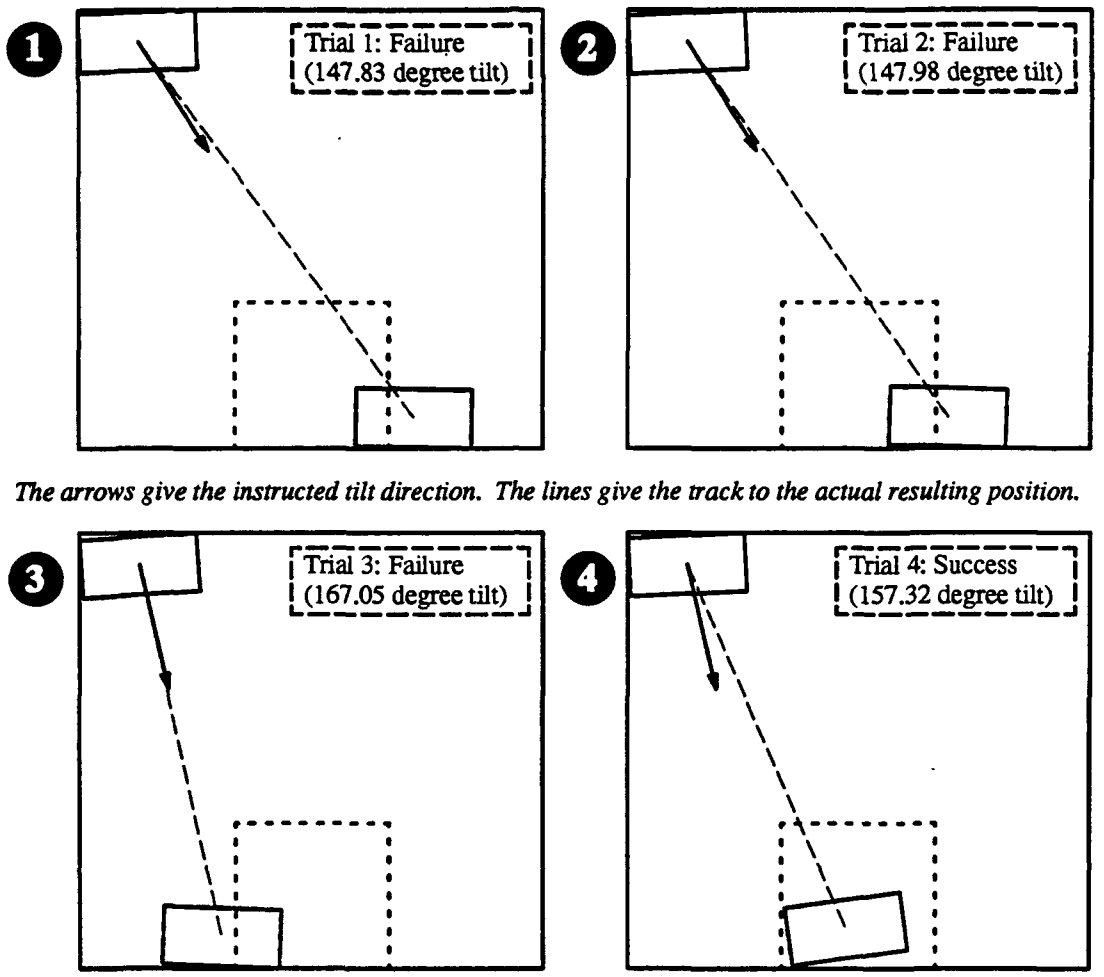


Figure 5.6. Refinement of a One-tilt Permissive Plan for Achieving the South Horizontal Configuration from the Northwest Horizontal Initial State

tray or the robot manipulating the tray which are beyond the approximate model employed by the permissive planner, illustrating how the stochastic planner adapts to the environment. Figure 5.6 shows a progression of four trials for the permissive planner after which the remaining 16 trials for this particular problem all succeed. A permissive plan is constructed making use of the approximate domain theory which includes approximately 250 rules and covers the basic physics of tray manipulation focusing on the frictional forces involved. Many important factors including mass of the block, friction between the tray and block, steepness of tilt of the tray, and direction of tilt of the tray are all somewhat uncertain and only roughly approximated. Due to different ways in which the block may slide to achieve the goal and due to the size of the goal sector, a range of tilts is available which

will achieve the goal. The initial permissive plan chooses tilts of about 148 degrees on the first two trials both of which fail, resulting in the block in the southeast sector but in the correct orientation. These two trials are depicted, respectively, in Figure 5.7 and Figure 5.8 . In these trials, the threshold for tuning is met after the second consecutive failure. Refinement then occurs to increase the tilt direction angle. This parameter is increased to the maximum which will still achieve the goal. This results in a tilt of 167 degrees which is shown in Figure 5.9. This also fails and tuning proceeds to decrease the angle to 157 degrees. This tilt action is shown in Figure 5.10. This trial results in the first of a series of 17 successes over the 20 repetitions. Notice that the final locations of the block as indicated do not necessarily correspond to the theoretical direction of the tilt. This is true of the success cases as well. Since the goal has been achieved, no further effort is expended in reducing the difference between the sensed final state and the projected final state. The empirical results presented in the next section involve many trials such as the one just illustrated.

5.4. Experimental Results

A comparison was performed between one-tilt permissive plans and one- and three-tilt optimal stochastic plans. The optimal stochastic plans were generated using the technique described in Section 5.1 The stochastic transition matrices were compiled using 3000 training examples. Generation of each optimal three-tilt stochastic plan is expensive and involves about 20,000 floating-point multiplications. A set of 52 problems was considered where the goal could be achieved in a single tilt while preserving block orientation. The permissive planner was given the specification of block location as returned by the vision system. Since the permissive planner is not limited to fixed matrix structures such as the stochastic approach, continuous quantities can be employed.

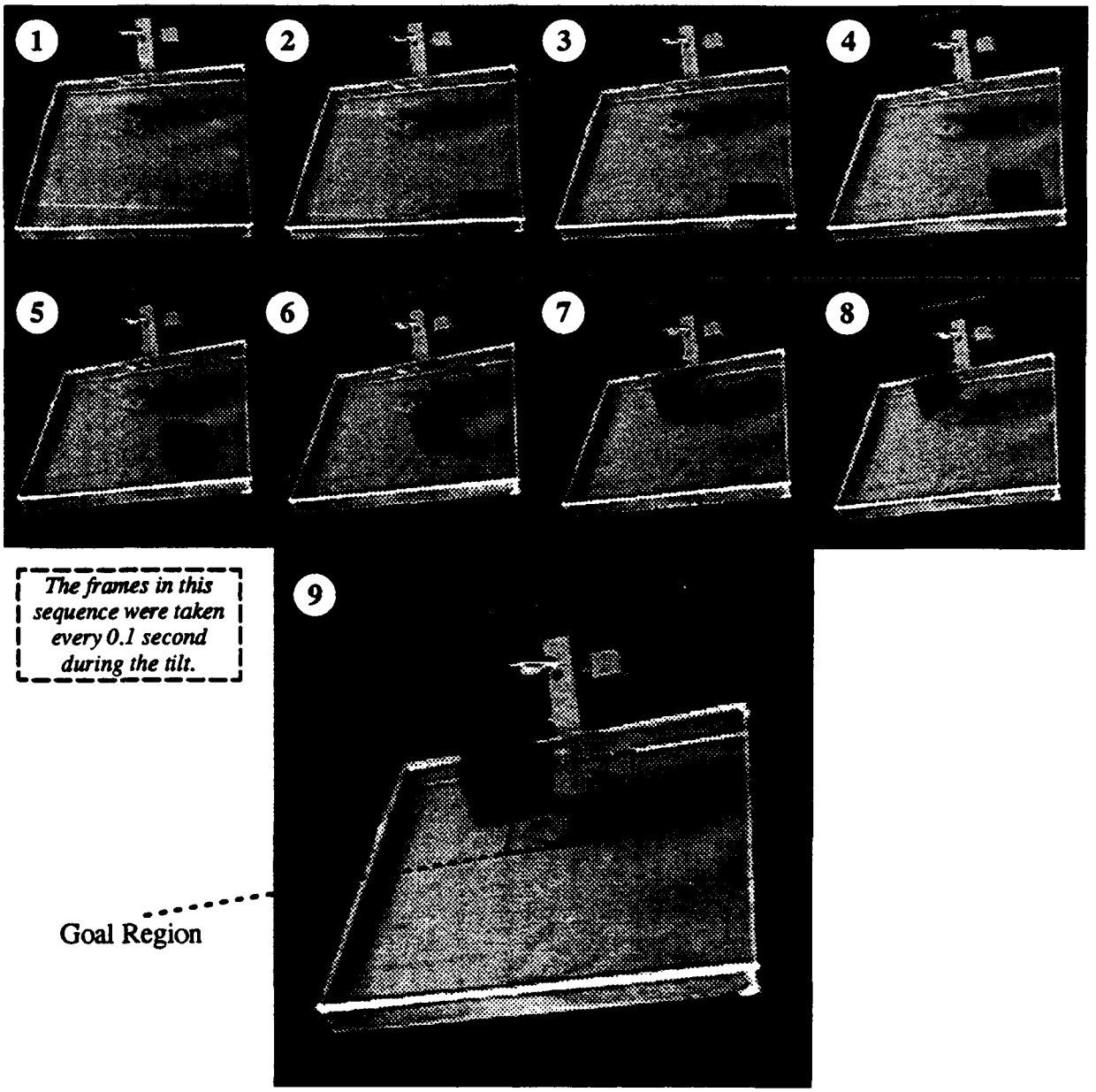


Figure 5.7. Permissive Plan Tilt of 147.83 Degrees Fails

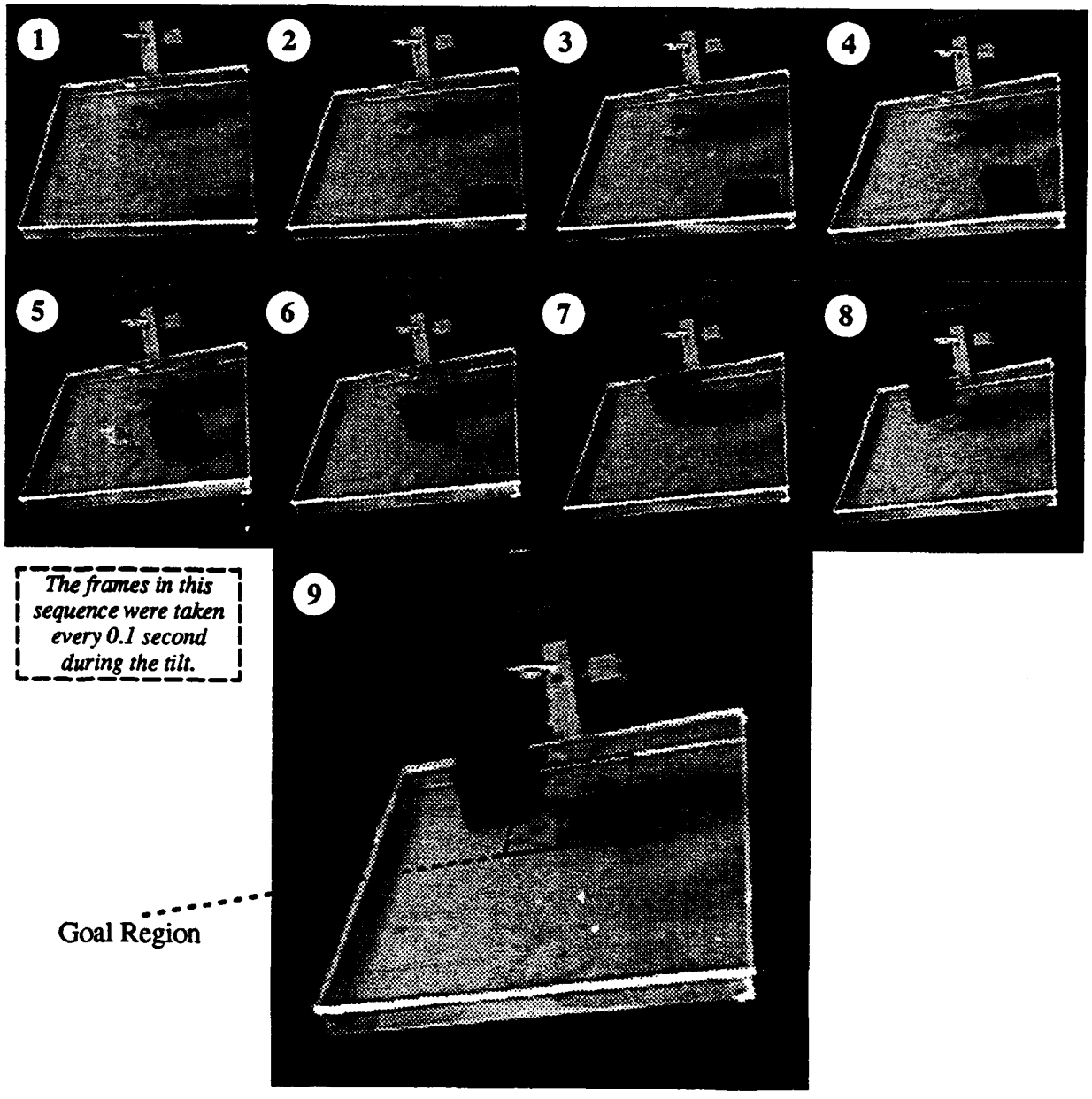


Figure 5.8. Permissive Plan Tilt of 147.98 Degrees Fails

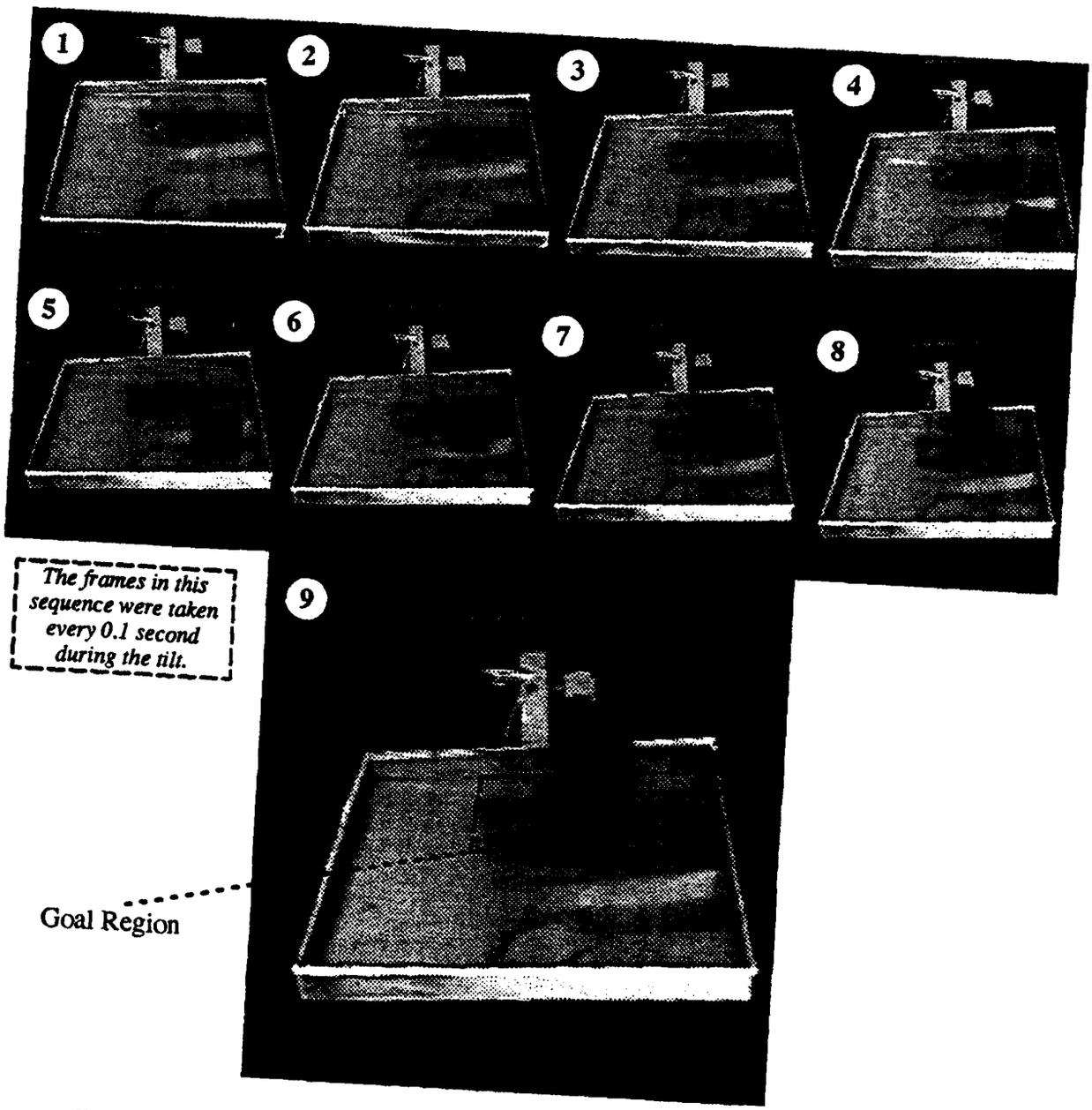


Figure 5.9. Permissive Plan Tilt of 167.05 Degrees (After First Tuning) Fails

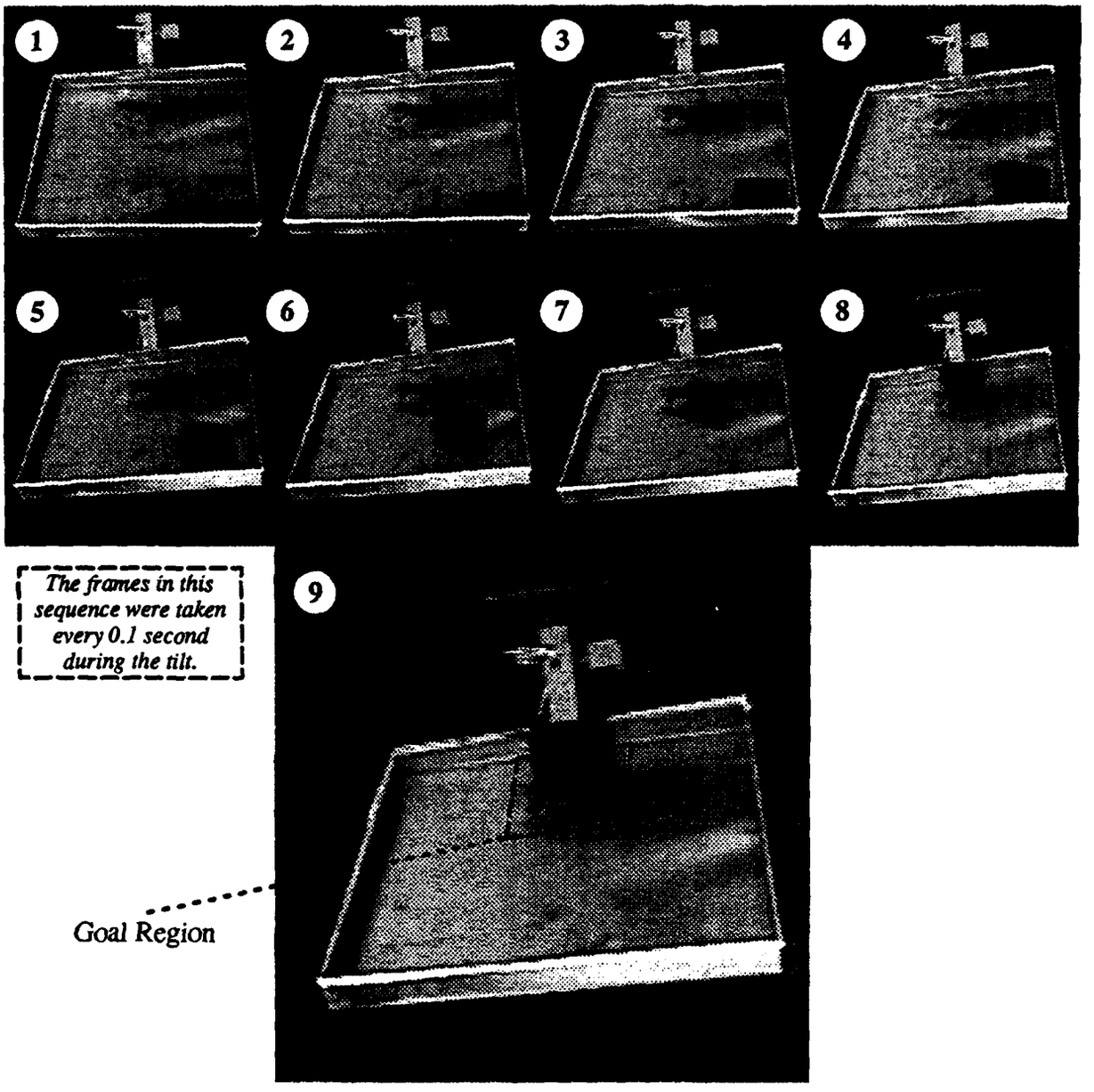


Figure 5.10. Permissive Plan Tilt of 157.32 Degrees (After Second Tuning) Succeeds

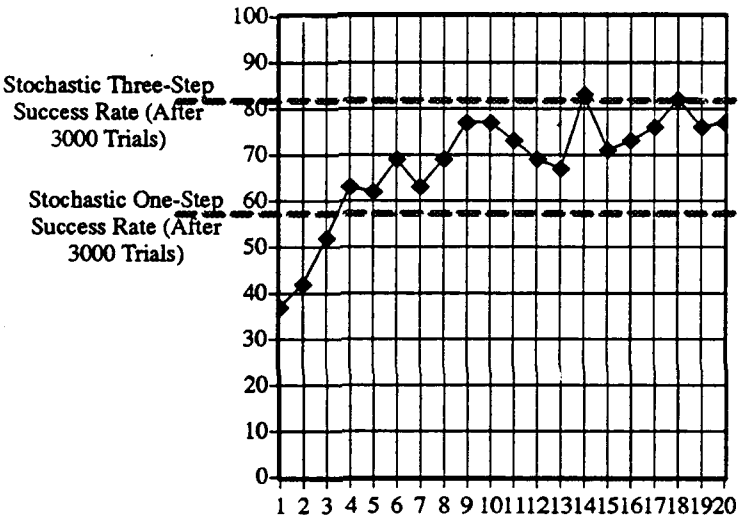


Figure 5.11. Two Traces of Average Success Rate for One-Step Permissive Plans For 52 Problems Over 20 Trials

In our first experiment, each of the 52 problems was given to the permissive planner 20 times (1040 training examples in all). Failures during those 20 repetitions resulted in tuning of the failed plans.[14] Figure 5.11 shows the average success rate of the problem set for each of the 20 repetitions. The tuning results in an increase from 37% success to 70-80% success. Significant intermediate dips in the success rate for particular problems are due to the policy of tuning to the extreme in the absence of additional information. If these "overshoots" cause failures, they are compensated for in the next refinement. The one- and three-tilt stochastic planner performance levels are superimposed on the graph. The single-tilt permissive plans perform significantly better than the single-tilt stochastic plans and result in a level of performance similar to that of the three-tilt stochastic plans. This performance is quite good considering the ability of the three-tilt stochastic plans to take advantage of multiple tilts to reduce errors in block orientation.

14. Since there may be several different qualitative distinctions of block configuration within one of the discrete states there may actually be several plans generated and tuned for the same problem.

In a second experiment, we obtained estimates of error on the results by averaging 20 runs of seven representative problems solved 20 times each. The averaged results are shown in Figure 5.12. The markers above and below the data points are the 95% accuracy error bars as determined by a T-test.

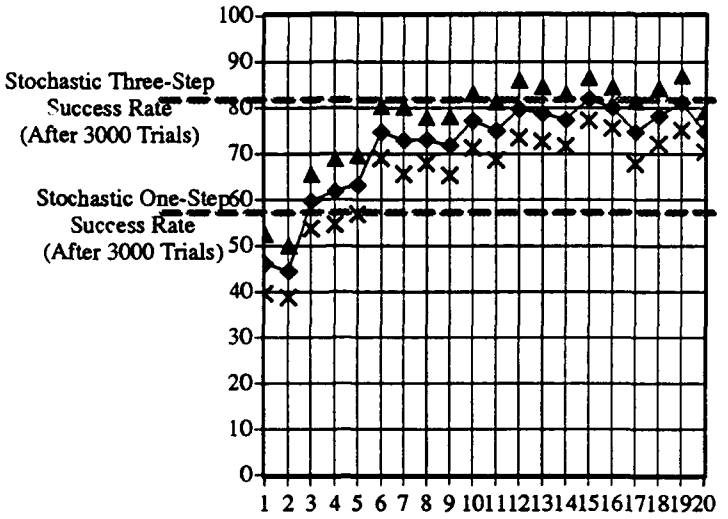


Figure 5.12. 20 Trials Each of One-Step Permissive Plans for 7 Problems Over 20 Trials

Permissive plans provide a method for achieving real-world goals using a domain theory to perform projection. This approach produces generalized plans with an explicit context, makes use of an approximate theory to provide tractability, can learn a successful set of plans for a problem with only a few training examples, and achieves a level of success similar to that of three-tilt stochastic plans. Along with the benefits of permissive planning comes the need for a domain theory, the assumption that failures are attributable to bad explicit approximations employed in the theory, and the need for the theory to be flexible enough such that parameters occur giving the planner multiple ways of accomplishing the goal.

6. RELATED WORK

In this chapter we discuss related work in a number of areas. First, we will discuss several approaches to planning for real-world domains. We follow with a comparison of permissive planning to a number of different approaches for learning to plan in complex domains. These include explanation-based techniques, stochastic and Bayesian approaches to solving real-world problems, inductive approaches, reactivity, neural networks, and case-based approaches.

One approach to planning in complex domains has been to pursue classical planning using a traditional AI micro-world theory. The hope here has been that a micro-world theory can be created that is "good enough" to allow the system to function in the real world. In the most straightforward version, the system implementor takes on the responsibility for insuring that no problems will result from necessarily imprecise descriptions of the domain. In general, this requires the implementor to characterize in some detail all of the future processing that will be expected of the system. Often he/she must anticipate all of the planning examples that the system will be asked to solve. If the physical robot system is not up to the accuracy that the examples require, the implementor must build a better vision system or purchase a more precise, more reproducible robot manipulator. This approach has enjoyed great popularity. While it is most often used in systems which research phenomena other than uncertainty, the implementors seldom more than tacitly acknowledge the implications. When employed by practical systems, the approach results in a never-ending quest for increasingly exacting (and expensive) hardware. The great irony of industrial automation, where this approach is nearly universal, is that the mechanical positioning capabilities of robots must far exceed the humans that they replace. The characteristic brittleness and inflexibility of industrial robotics is a consequence of the presumption that the implementor can anticipate all future applications.

A second approach involves manipulating explicit representations of uncertainty [Brooks82, Brost88, Davis86, Erdmann86, Hutchinson90, Lozano-Perez84, Zadeh65]. This approach attempts to model uncertainties and thus preserves the classical planning ideal of a provably correct plan so long as the uncertainty models are correct. Unfortunately, a high computational price is incurred. A general ability to project states including objects with explicit general error bounds is necessarily no less difficult than if the objects are known precisely. Many systems incorporate simplifications, typically assuming that uncertainties are constant, independent of context, or otherwise constrained. This buys some efficiency at the price of generality. But the efficiency is never greater than if the objects had zero uncertainty.

A third approach to planning is one commonly used when handling spatial uncertainties in robotics. In this approach, in a sense, a kind of worst-case representation is assumed for objects in the world. The approach, which seems only to be used for problems of path planning, includes techniques such as quantizing the space [Malkin90, Wong85, Zhu90] and imagining a repulsive potential field around obstacles [Hwang88, Khatib86]. Interestingly, the approach can be more efficient than for the zero uncertainty case. Since object boundaries are not guaranteed to be the tightest possible, they can be selected to be both conservative and simplifying. This benefit does not come without cost. In different guises, completeness or correctness can be sacrificed. In a sense, this is the closest of the popular approaches to our research. In a sense we also adopt a conservative representation, although the uncertainty tolerance is due to plan characteristics rather than explicit representations. This shift supports a context-sensitive conservatism which supports reasoning about general manipulation problems rather than only path planning in a static world.

In an attempt to circumvent the tractability/performance tradeoff of classical planners, many researchers pursued reactivity [Agre87, Firby87, Schoppers87, Suchman87]. Instead of projecting the effects of actions, these techniques sense the state of the world and then react by performing actions. A purely reactive system essentially uses the real world to discover the effects of actions rather

than employing a domain theory as with the classical planner. One shortcoming of purely reactive systems is that they lack precisely what classical planners offer: long-term goal-directedness. They are good at responding locally to a situation but lack the guidance to compose local actions into a good overall plan. Some recent research has addressed ways to combine classical and reactive planning to address this shortcoming [Cohen89, Drummond90, Firby87]. Nevertheless, reactive techniques place heavy demands on sensing. The time it takes to gather and process sensory data can compromise execution speed. Adding to this is the problem of choosing the correct sensors to consider. Any system which operates real-world devices, like mobile robots, has significant amounts of sensory data available: too much to completely process in real time. Reactive systems too must use their reactive rules to focus attention on particular sensors. Achievement of the desired performance levels depends on how carefully one crafts the reactive rules. Reactive planning holds much promise but many open problems remain to be solved.

In addition to the planning approaches above, many different approaches have been pursued which utilize machine learning for improving problem solving in complex domains. The first group of these systems, such as classical planning systems, employ micro-world theories in conjunction with learning techniques. One early system, STRIPS, controlled a mobile robot [Fikes72]. It was able to learn macro-operators for operating a robot from analysis of its own problem solving. The generalization technique used for constructing the macro-operators is very similar to the EGGS technique used in explanation-based learning today [Mooney86].

The first work in demonstrating the application of explanation-based learning (EBL) [DeJong86, Mitchell86] to problem solving in robotics was done by Segre in his ARMS system [Segre87]. ARMS observes a human operator achieving some goal through sending commands to a robot manipulator. Using its domain theory, the system is able to construct a general plan for achieving the goal. The plan is not sensitive to incidentals in the original training example as are plans generated

using other learning techniques, because only aspects of the training example which support the goal, according to the domain theory, find their way into the final plan.

Both STRIPS and ARMS work in the inherently complex domain of robotics and yet do so with simple fixed micro-world theories. However, since the world is never ideal like the model, systems such as these have no way of reasoning about possible disparities between the model and real world. Learned plans are saddled with the representational shortcoming of the original theory with no refinement to overcome difficulties.

One approach was to follow those in classical planning who would construct more and more complicated theories so as to reason about and overcome difficulties encountered in the real world. Techniques such as explanation-based learning require explanations to be constructed from those ever more complicated theories. To arrive at a reusable general plan, the system might be required to tackle an intractable planning problem. The proposed solution was to send a different sort of machine learning to the rescue: learn what parts of the theory can be approximated to make the theory tractable for planning. Examples of this work include both Keller's METALEX [Keller87] and Zweben's [Zweben88] work in dropping preconditions to domain rules when the effectiveness of the rules can still be maintained as determined empirically. Mostow and Fawcett in the HDF system [Mostow87] and Ellman in his POLLYANNA system [Ellman88] characterize approximation as a search in a space of approximate theories for an approximation meeting certain desired criteria. Others like Chien employ assumptions which are made and retracted in order to promote tractable planning and learning [Chien90]. In permissive planning, our domain theory corresponds to one of these approximate theories because it is based on explicit approximations. Permissive planning is not concerned with changing the approximation (or set of assumptions) but with refining the plan to work in spite of them. While the time efficiency/accuracy tradeoff is a primary motivation of many of the systems employing approximations, we focus on the need for improving the success rate of plans in complex, real-world domains. Permissive plan refinement makes the plans more tolerant of in-

adequacies in the representation. The approaches are complementary in that time efficiency, accuracy, and tolerance of representational inadequacies are all important aspects of a system's real-world performance.[15]

In working with real-world problems, many things are known to be approximate and should be represented as such. It makes little sense to "discover" that a block's location is really approximate, e.g., is uncertain. Simply declaring something as approximate does not degrade a system's performance unless approximations are considered explicitly in the reasoning conducted to carry out some actions. However, approximations do provide an important mechanism for guiding plan refinement if a system is not performing to expectations. At that time, plans are refined to work in spite of the poor approximations.

A different tack was taken by Minton in his PRODIGY system [Minton87]. He focuses on speed improvements that can be gained by learning better control knowledge for the search process. A naive planner is used to construct a plan for achieving some goal. The planner makes implicit assumptions, for example, about the independence of subgoals. After a proof structure for achieving the goal has been constructed (or while it is being constructed for in-trial learning), observed failures are generalized into control rules which prevent the system from making unwise choices during such a proof. The system learns methods by which a search can be made more efficient. This can be very useful but the system cannot effectively deal with intractability unless it has the ability to make and assess explicit approximations. Without such a method, PRODIGY can easily be overwhelmed with the size of search space.

Motivated by shortcomings of PRODIGY, Gratch has introduced a statistically sound way of evaluating competing efficiency transformations in his COMPOSER system [Gratch91, Gratch92]. We

15. For a model of operationality for real-world systems which integrates several important factors see [Bennett89].

use a similar statistical evaluation technique in our algorithm of Chapter 2 for the purpose of improving the probability of success and coverage of the permissive plan at each refinement step.

Altogether, the techniques for learning to plan efficiently discussed above help to make less tractable problems more tractable. They allow a system to tractably employ a more sophisticated theory. Yet, they do not attack the root problem for real-world domains, which is that any discrepancy between the theory and the world can result in unwanted failures. A permissive planner utilizes feedback from interaction with the world which is essential.

Another approach to learning in complex domains is the stochastic planning technique discussed in Chapter 5. The system of Christiansen and Goldberg [Christiansen90] needs a large number of examples to establish probabilities. It also quickly becomes intractable as the number of discrete world states that must be distinguished grows. The lack of a theory also limits performance as compared with the permissive planner as was seen in the experimental trials of Chapter 5.

Dean has a system based on Bayesian decision theory which utilizes a specialization of a Bayesian network,[16] called a temporal belief network, for controlling a mobile robot on a journey through a series of rooms [Dean90]. The topology for the network is designed in advance for the task. As with the stochastic approach, states are discretized to make the solution tractable. This limits the resolution at which the system can sense and act in the world. The permissive planning approach can employ continuous quantities tractably and does not suffer this drawback. The Bayesian network approach, like the stochastic approach, also requires gathering enough examples to calculate initial probabilities for the network. With permissive planning, because a theory is employed, only a single example is needed at each stage of refinement. It should be pointed out that, although elements of the algorithm may be expensive, decision theory, in general, has the advantage of allowing prediction of the complexity of various operations and may facilitate decisions to avoid operations deemed

16. For an excellent discussion of Bayesian networks see [Pearl88].

"too expensive." However, here, it can only help in deciding among alternatives within the Bayesian network framework adopted by the approach. In permissive planning, our primary goal is the desired balance between probability of success for a plan and coverage. Had we additional goals as to the desired efficiency of refinement along with some knowledge about the likelihood of a positive payoff for some particular refinement, decision theoretical techniques might be applied to guide refinement choices.

Another common approach for complex domains has been to pursue purely inductive learning techniques where one utilizes a multitude of examples to substitute for an initial domain theory. Mason, Christiansen, and Mitchell, whose tray-tilting work was discussed in Chapter 5, performed experiments with inductive learning and tray tilting in 1989 [Mason89]. Their inductive agent used a version-space approach to learn the bounds on tilt angle ranges which led to goal achievement. They observed that performance improvement occurred after about 20 trials and leveled off around one-hundred trials. The agent converged to about 50% success. Among their conclusions was that additional analytical knowledge should be investigated as the inductive approach represented only a very weak theory of tray-tilting. Consequently, the learning rate was slow and the final achieved success rate was disappointing (compared to their human-devised theory with 95% success). Permissive planning presents a positive alternative to this approach requiring few training examples, using commonly available analytical knowledge about the domain, and achieving far better than a 50% success rate.

Yet another approach has been to pursue neural network learning for complex domains. Mel's MURPHY system exemplifies such approaches and is a first step in learning using neural networks to aid in a robot manipulation task [Mel88]. MURPHY uses knowledge about a robot arm's current joint configurations in conjunction with visual data about the joint configurations to learn connections between the two. This can be accomplished without an intelligent teacher by having the robot

step through a representative sample of the 1 billion possible joint configurations. MURPHY can use its learned connections to "envision" sequences of actions for planning.

MURPHY is quite appealing as a method for learning sensory-motor interaction in a robot arm because of its simplicity. It requires little domain knowledge but faces the common disadvantages of such neural network techniques: it requires a myriad of examples, needs an environment to "train" with where failures have negligible cost, and with little domain knowledge can be sensitive to incidental associations made from observations. Furthermore, permissive planning produces general declarative plans with a clear interpretation. In MURPHY, the result is a network that exhibits a particular behavior but is difficult to analyze.

Another approach for planning in complex domains is to use a case-based approach. Hammond's CHEF system [Hammond86] illustrates how a case-based planner learns from previous failures. When the system encounters a situation in which the plan it developed fails, it indexes the failure under a generalized set of features which indicate why the failure occurred as well as a set of features which help to predict the failure. During future planning, the system uses the failure predictive features to focus on avoiding similar failures during the construction of the new plan.

Central to the CHEF system is the notion of a powerful case-based planner that can select relevant failures and incorporate fixes for them into the current plan. Furthermore, the possible plan fixes have to be selected from a fixed set already coded into the system. Permissive planning, although restricted by the domain theory provided, develops tuning hypotheses from parameters found by the system in the theory. Early case-based systems such as CHEF operated in micro-world domains and did not face the difficulties inherent in complex real-world domains. However, second-generation case-based systems were applied to complex real-world problems. These include systems such as FIRST [Daube89], which worked in the domain of structural beam design, ROENTGEN [Berger91], which designs radiation therapy procedures, and Pandya's case-based motion planner

[Pandya92]. All of these systems work with real numeric constraints and data. ROENTGEN, for instance, which addresses uncertainty, seems to use numeric uncertainty bounds not unlike the uncertainty modelling approaches. All of the approaches share a common general approach of retrieving similar plans, modifying those plans, and continuing to modify them when failures are observed. Nevertheless, the real-world systems use different techniques for retrieval and repair often based on particular characteristics of the domain. In contrast, permissive planning is a domain-independent planning technique for complex, real-world domains. Recently, there has been an increasing emphasis on cross-domain case-based techniques. Case-based reasoning continues to be the topic of much current research.

7. CONCLUSIONS

7.1. Contributions and Requirements of Permissive Planning

In this thesis we presented the new technique of permissive planning which, although not universally applicable, offers a powerful new approach to planning in many complex, real-world domains. A number of requirements must be met in order for permissive planning to be employed. Permissive planning employs an explicitly stated theory of the domain. This is the same requirement which must be met by classical planning systems. However, domains do exist which are less well-understood where rules of behavior are difficult to construct. Such domains lend themselves to inductive rather than deliberative approaches such as permissive planning.

It also must be recognized that, although the domain theory gives much useful planning information, it necessarily colors the resulting permissive plans. The way knowledge is represented in the domain theory, for instance, affects the parameters which are available to the permissive planner in refinement. In the latter grasping example of Chapter 4, one parameter was the clearance of the gripper in surrounding the object. The permissive planner would not have been able to recognize "clearance" as a parameter if it were not for the explicit representation of "clearance" in the theory. The explicit representation of "clearance" establishes the importance of the quantity representing the difference between the chosen gripper opening width and the width of the object.

The theory may also impede permissive planning if it is too conservative and seeks a nearly guaranteed solution. Such a theory is likely to be computationally expensive to employ. Furthermore, it may often be unable to produce a plan for a given situation. This gives the permissive planner little opportunity to execute and refine plans. It should be recognized, however, that there is a place for conservative, guaranteed planners when failures cannot be tolerated. In critical domains, such as the control of a nuclear plant, the extra computational effort may be well worth the cost of a failure.

Permissive plan refinement requires that explicit approximations be employed in the plan's supporting explanation structure. Each failure hypothesis is tied to a hypothesized failing explicit approximation. It is required that explicitly approximate quantities will be more likely to have small rather than large deviations from the unknowable true value for the quantity.

Permissive planning uses a refinement process triggered by failures. Specifically, our implementation employs monitored actions to detect failures during plan execution and to associate them with the support proof for the failed expectations. A permissive planner is more likely to arrive at successful plans quickly in domains where more specific failure information is available. Failure information helps in determining the plausibility of different failure hypotheses.

A number of benefits are offered by the permissive planning approach which make it worthwhile to consider for many real-world planning problems. As we have discussed above, it is necessary to provide a domain theory for permissive planning. However, with permissive planning, the domain theory is able to provide a large amount of power without making the same sacrifices as do classical approaches. Inductive learning approaches plateau at a limited success level in complex domains without the incorporation of that knowledge. Classical planning techniques relying on projection using a theory of the domain lack robustness in real-world domains, when discrepancies between the theory and the world become important. Permissive planning seeks to employ theory without a commensurate reduction in real-world robustness.

Permissive planning operates very much in an "on-demand" manner. Explicit approximations are not considered while planning only during refinement, thus making permissive planning more efficient at planning time than approaches which model and propagate uncertainties. It is also not necessary to commit to a specific uncertainty model or error bounds in advance as with the other approaches.

Only observed failures are addressed by a permissive planner. Resulting permissive plans are keyed through experience and refinement to the distribution of problems presented to the system. A savings in time efficiency is gained over planners which universally seek to reduce failures independent of the problem distribution. Furthermore, a successful permissive plan may be found to cover a given problem distribution which might not have been considered by a planner seeking to reduce failures in general.

In many real-world domains it can become quite expensive to perform sensing of the world and to process that information. Permissive planning offers a means for reducing the sensing requirements through increased use of projection using the domain theory with the permissive plan refinement process to remedy discrepancies with the world.

Unlike systems that re-plan everything each time they are asked to achieve a goal, the permissive planner learns general plans for a class of situations. The plans carry preconditions that describe in what situations they apply. Such general plans embody the refinements of earlier learning trials bringing them to new but similar situations.

The permissive planning algorithm offers a rigorous statistical basis for evaluating refined plans. A target success rate and coverage can be defined which the planner seeks to satisfy. The ability to specify such a target criteria is ideal for complex domains where uncertainty plays a role.

We have presented a basic theory of permissive planning but have gone further to illustrate how a practical approximation to that theory has been implemented. The results in the robotic grasping and tray-tilting domains have given a strong demonstration of the viability of the approach. Yet, many promising future research directions remain.

7.2. Future Work

Let us discuss some future directions for this work. Permissive planning as presented in this thesis finds a single region in the space S_{GP} where the plan will achieve some desired probability of success and coverage. It may well be that for a given distribution of problems, a disjunctive description may yield a better result. Our current framework could use multiple permissive plans to yield a better result, each plan corresponding to one of the disjuncts. However, how can "appropriate" disjuncts be discovered? We believe that inductive learning techniques could be applied to yield the disjuncts using failures of the original plan as examples. For instance, in the tray-tilting domain rough spots in the tray cause difficulties in achieving part orientation goals. Imagine that a rough spot exists on the tray bottom such that it affects the slide of a block along one of the tray diagonals but not the opposite diagonal. A domain theory exploiting tray symmetry for planning would treat the two diagonals as equivalent. In one random problem distribution it may be that the two diagonals are used equally often but that one fails consistently. The permissive planner would try to refine the plan by tuning plan parameters. However, even the refined plan makes no distinction between the two diagonal slides employed. If the desired probability of success and coverage is not achieved, it makes sense to consider the failures of the plan using induction. If a distinction can be found which separates the failed uses of the plan from the successes, the distinction can be employed to create two separate plans. In this case, each of the two new plans would correspond to a single diagonal. In part, the permissive planner's failure to achieve the success criteria with the single plan might be due to suggested tuning hypotheses for each of the diagonals opposing each other. The compromise reached in refinement of the single plan yields an unsatisfactory success rate. With two plans, the failing one can now be refined independently.

In our permissive planner, approximations are explicitly labelled in the domain theory by the user. However, a domain-independent theory about error could be used to assist in identifying explicit approximations. Furthermore, the work on "when to approximate" might be brought to bear in the

permissive planner by deciding when a truly approximate quantity should be labelled as such or not. This would not affect the speed of the planner as explicit approximations are not considered during planning. However, efficiency gains in the speed of the refinement process could be achieved. For instance, if it were found that a particular parameter was frequently being entertained in the refinement process but was never selected for refinement, one could make refinement more efficient at the cost of eliminating some possible permissive plans by removing that particular explicit approximate label.

A permissive planner never guarantees that a permissive plan can be found, only that it seeks the desired success rate and coverage and will return a plan which satisfies them if possible. Permissive planning subscribes to the idea that the plan is refined to work in spite of discrepancies. Complementary to that goal is the need to remedy discrepancies when the permissive planner is unable to produce working plans. One interesting area for future work involves how to make an approximation of the world to facilitate efficient planning. A tradeoff naturally exists between accuracy of the model and time efficiency of planning using the model. A permissive planner should be able to trigger refinement of the model when it has exhausted other alternatives. Through a knowledge of the plans which use the model, the permissive planner's refinement process should be better able to refine the model than knowledge-poor model refinement techniques.

Permissive planning combined with other planning techniques for real-world domains could eventually be embedded into complex systems enabling automatic adaptation to the environment. Imagine a robotic manipulator and vision system that once unboxed and set up begins to improve at the manipulation task assigned it taking advantage of the distribution of tasks it encounters. Working in a environment with little contrast between the pieces and the background it fails to identify objects in the workspace but compensates by adjusting the threshold parameter in its vision system, not by demanding better lighting. It fails to move the end effector to the correct coordinate in the vision system as predicted by its model. It hypothesizes a belt slipping and slows down the speed of move-

ment for this joint configuration eliminating that particular failure. Later, it commands a joint to move, but it does not move as verified by the camera. The leading tuning hypothesis suggests choosing another member of a discrete set of comparable formats for the move command. The alternative move command consistently works and is now preferred. A useful working real-world system such as this is a long term goal of this work. It will no doubt require a collection of learning and planning techniques of which permissive planning is an important component.

APPENDIX. DETAILS OF THE *PERMPLAN* ALGORITHM

Below are more detailed listings of the procedures employed in the permissive planning algorithm presented in Chapter 2.

***Execute**(GP) Function:*

{This function executes general plan GP on some $\vec{V} \in$ Projected Region $\subseteq S_{GP}$ to achieve its goal, returning a 2–tuple of $< success, \vec{V} >$ on success and $< failure, \vec{V} >$ on failure.}

***SignWithDelta1**(Util(i), δ , ETR) Function:*

$n \leftarrow 0$	*{iteration variable}*	*(1)*
repeat		*(2)*
$n \leftarrow n+1$	*{increment example number}*	*(3)*
$EX \leftarrow$ ***Execute**(GP)*	*{execute GP on problem from projected region}*	*(4)*
$X_n \leftarrow$ ***PSC**(EX,ETR)*	*{return psc rating}*	*(5)*
$<done, \overline{Util_n}> \leftarrow$ ***Nádas**(Util(i), n, δ, 1)*	*{check stopping criterion}*	*(6)*
until *done*		*(7)*
return $\overline{Util_n}$		*(8)*

***SignWithDelta2**(Util(i), δ , CurrentETR, R) Function:*

$n \leftarrow 0$	*{iteration variable}*	*(9)*
repeat		*(10)*
$n \leftarrow n+1$	*{increment example number}*	*(11)*
$EX \leftarrow$ ***Execute**(GP)*	*{execute GP on problem from projected region}*	*(12)*
$X'_n \leftarrow$ ***PSC**(EX,CurrentETR)*	*{return psc rating of CurrentETR}*	*(13)*
$X_n \leftarrow$ ***PSC**(EX,R)*	*{return psc rating of R}*	*(14)*
$<done, \overline{Util_n}> \leftarrow$ ***Nádas**(Util(i), n, δ, 1)*	*{check stopping criterion}*	*(15)*
until *done*		*(16)*
return $\overline{Util_n}$		*(17)*

***PosWithDelta1**(Util(i), δ , ETRs) Function:*

```
n ← 0                                          {initialize example number}                    (1)
GoodETRs = {}                                  {initialize ETRs with positive utility}        (2)
BadETRs = {}                                   {initialize ETRs with negative utility}        (3)
repeat                                                                                        (4)
   n ← n+1                                     {increment example number}                     (5)
   EX ← Execute(GP)                            {execute GP on problem from projected region}  (6)
   j ← 0                                       {initialize transformation counter}            (7)
   foreach ETR in ETRs                                                                        (8)
      j ← j+1                                                                                 (9)
      if ETR ∉ (GoodETRs ⋃ BadETRs) then                                                      (10)
      begin                                                                                   (11)
         X_{n,j} ← PSC(EX,ETR)                 {give PSC rating for NewETR}                   (12)
         <done, Util_n> ← Nádas(Util(i), n, δ, |ETRs|)  {check stopping criterion}           (13)
         if done then                                                                         (14)
            if Util_n ≥ 0 then                                                                (15)
               GoodETRs = GoodETRs ⋃ {ETR}                                                    (16)
            else                                                                              (17)
               BadETRs = BadETRs ⋃ {ETR}                                                      (18)
      end                                                                                     (19)
   endforeach                                                                                 (20)
until ((ETRs–GoodETRs–BadETRs) = ∅)                                                           (21)
return GoodETRs                                                                               (22)
```

PosWithDelta2*(Util(i), δ , NewETRs, CurrentETR) Function:*

```
n ← 0                                          {initialize example number}                    (1)
GoodETRs = {}                                  {initialize ETRs with positive utility}        (2)
BadETRs = {}                                   {initialize ETRs with negative utility}        (3)
repeat                                                                                        (4)
   n ← n+1                                     {increment example number}                     (5)
   EX ← Execute(GP)                            {execute GP on problem from projected region}  (6)
   j ← 0                                       {initialize transformation counter}            (7)
   foreach NewETR in NewETRs                                                                  (8)
      j ← j + 1                                                                               (9)
      if NewETR ∉ (GoodETRs ⋃ BadETRs) then                                                   (10)
      begin                                                                                   (11)
         X_{n,j} ← PSC(EX,NewETR,w_c,w_s)         {give PSC rating for NewETR}                (12)
         X_n ← PSC(EX,CurrentETR,w_c,w_s)         {give PSC rating for CurrentETR}            (13)
         <done, Util_n> ← Nádas(Util(i), n, δ, |NewETRs|)   {check stopping criterion}        (14)
         if done then                                                                         (15)
            if Util_n ≥ 0 then                                                                (16)
               GoodETRs = GoodETRs ⋃ {ETR}                                                    (17)
            else                                                                              (18)
               BadETRs = BadETRs ⋃ {ETR}                                                      (19)
      end                                                                                     (20)
   endforeach                                                                                 (21)
until ((NewETRs–GoodETRs–BadETRs) = ∅)                                                        (22)
return GoodETRs                                                                               (23)
```

Nádas*(Util(i), n, δ, n_d) Function:*[17]

$$\overline{Util_n} \leftarrow \frac{\sum_{i=1}^{n} Util(i)}{n} \qquad \{compute\ average\}$$

$$S_n^2 \leftarrow \frac{1}{n}\sum_{i=1}^{n}\left(Util(i) - \overline{Util_n}\right)^2 \qquad \{compute\ variance\}$$

$$Let\ \Phi(a) = \frac{\delta}{2n_d}\ where\ \Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy$$

if $\left(\frac{S_n^2}{[\overline{Util_n}]^2} \le \frac{n}{a^2}\right)$ **and** $(n \ge 3)$ **then**

   return <**true**, $\overline{Util_n}$>

else

   return <**false**, $\overline{Util_n}$>

17. This stopping criterion is a special case of the proportional accuracy criterion described in [Nadas69].

PSC(<outcome, $\vec{V}$ >,ETR) Function:

if $\left(\vec{V} \in ETR\right)$ **or** *(outcome=failure)*
  $C \leftarrow 1$ *{GP covered example EX}*
else
  $C \leftarrow 0$ *{GP did not cover example EX}*
if $\left(\vec{V} \in ETR\right)$ **and** *(outcome=success)*
  $S \leftarrow 1$ *{GP succeeded}*
else
  $S \leftarrow 0$ *{GP failed or did not apply}*
return $\left(w_c C + w_s S\right)$

REFERENCES

[Agre87] P. Agre and D. Chapman, "Pengi: An Implementation of a Theory of Activity," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 268–272.

[Bennett89] S. W. Bennett, "Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains," M.S. thesis, ECE, University of Illinois, Urbana, IL, January 1989. (Also appears as Technical Report UILU–ENG–89–2204, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign)

[Berger91] J. Berger and K. Hammond, "A Memory–Based Approach to Radiation Therapy Treatment," *Proceedings of the Case–Based Reasoning Workshop*, May 1991, pp. 203–214.

[Brooks82] R. A. Brooks, "Symbolic Error Analysis and Robot Planning," Memo 685, MIT AI Lab, Cambridge, MA, September 1982.

[Brost88] R. C. Brost, "Automatic Grasp Planning in the Presence of Uncertainty," *The International Journal of Robotics Research 7*, 1 (February 1988), pp. 3–17.

[Chien90] S. A. Chien, "Incremental Reasoning for Planning and Explanation–based Learning," Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, IL, November 1990.

[Christiansen] A. D. Christiansen, M. T. Mason and T. M. Mitchell, "Learning Reliable Manipulation Strategies without Initial Physical Models," *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pp. 1224–1230.

[Christiansen90] A. D. Christiansen and K. Y. Goldberg, "Robotic Manipulation Planning with Stochastic Actions," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 3–8.

[Cohen89] P. R. Cohen, M. L. Greenberg, D. M. Hart and A. E. Howe, "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments," *Artificial Intelligence Magazine 10*, 3 (1989), pp. 32–48.

[Daube89] F. Daube and B. Hayes–Roth, "A case–based mechanical redesign system," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 50–56.

[Davis86] E. Davis, *Representing and Acquiring Geographic Knowledge*, Morgan Kaufmann Publishers, Inc., 1986.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation–Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145–176. (Also appears as Technical Report UILU–ENG–86–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Dean90] T. Dean, K. Basye and M. Lejter, "Planning and Active Perception," *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, San Diego, CA, November 1990, pp. 271–276.

[Drummond90] M. Drummond and J. Bresina, "Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, August 1990, pp. 138–144.

[Ellman88] T. Ellman, "Approximate Theory Formation: An Explanation–Based Approach," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 570–574.

[Erdmann] M. Erdmann and M. T. Mason, "An Exploration of Sensorless Manipulation," *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pp. 1569–1574.

[Erdmann86] M. Erdmann, "Using Backprojections for Fine Motion Planning with Uncertainty," *International Journal of Robotics Research 5*, 1 (1986), pp. 19–45.

[Fikes71] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence 2*, 3/4 (1971), pp. 189–208.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251–288.

[Firby87] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 202–206.

[Gratch91] J. Gratch and G. DeJong, "A Hybrid Approach to Guaranteed Effective Control Strategies," *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL, June 1991.

[Gratch92] J. Gratch and G. DeJong, "COMPOSER: A Probabilistic Solution to the Utility Problem in Speed–up Learning," *Tenth National Conference on Artificial Intelligence*, San Jose, CA, 1992.

[Hammond86] K. Hammond, "CHEF: A Model of Case–Based Planning," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 267–271.

[Hutchinson90] S. A. Hutchinson and A. C. Kak, "Spar: A Planner That Satisfies Operational and Geometric Goals in Uncertain Environments," *Artificial Intelligence Magazine 11*, 1 (1990), pp. 30–61.

[Hwang88] Y. K. Hwang, "Robot Path Planning Using Potential Field Representations," Ph.D. thesis, Electrical Engineering Department, University of Illinois, Urbana, IL, 1988.

[Keller87] R. M. Keller, "The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance," Ph.D. thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1987. (Also appears as Machine Learning Technical Report #7, Laboratory for Computer Science Research, Rutgers University)

[Khatib86] O. Khatib, "Real–Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research 5*, 1 (1986), pp. 90–98.

[Lozano–Perez84] T. Lozano–Perez, M. T. Mason and R. H. Taylor, "Automatic Synthesis of Fine–Motion Strategies for Robots," *International Journal of Robotics Research 3*, 1 (Spring 1984), pp. 3–24.

[Malkin90] P. K. Malkin and S. Addanki, "LOGnets: A Hybrid Graph Spatial Representation for Robot Navigation," *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, July, 1990, pp. 1045–1050.

[Mason89] M. T. Mason, A. D. Christiansen and T. M. Mitchell, "Experiments in Robot Learning," *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, June 1989, pp. 141–145.

[Mel88] B. W. Mel, "Building and Using Mental Models in a Sensory–Motor Domain: A Connectionist Approach," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 207–213.

[Minton87] S. Minton and J. Carbonell, "Strategies for Learning Search Control Rules: An Explanation–based Approach," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 228–235.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar–Cabelli, "Explanation–Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47–80.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation–Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551–555. (Also appears as Technical Report UILU–ENG–86–2216, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Mooney87] R. J. Mooney, "A General Mechanism for Explanation–Based Learning and Its Application to Narrative Understanding," Ph.D. thesis , Department of Computer Science, University of Illinois, Urbana, IL, 1987.

[Mostow87] J. Mostow and T. Fawcett, "Approximating Intractable Theories: A Problem Space Model," Machine Learning–Technical Report 16, Department of Computer Science, Rutgers University, New Brunswick, NJ, December 1987.

[Nadas69] A. Nadas, "An Extension of a Theorem of Chow and Robbins on Sequential Confidence Intervals for the Mean," *The Annals of Mathematical Statistics 40*, 2 (1969), pp. 667–671.

[Nilsson80] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980.

[Pandya92] S. Pandya, "A case–based approach to robot motion planning," M.S. thesis, Department of Computer Science, University of Illinois, Urbana, IL, August 1992.

[Pearl88] J. Pearl, in *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, San Mateo, CA, 1988.

[Press86] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1986.

[Schoppers87] M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 1039–1046.

[Segre87] A. M. Segre, "Explanation–Based Learning of Generalized Robot Assembly Tasks," Ph.D. thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, January 1987. (Also appears as UILU–ENG–87–2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign.)

[Suchman87] L. A. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, 1987.

[Taylor87] R. H. Taylor, M. T. Mason and K. Y. Goldberg, "Sensor–based manipulation planning as a game with nature," *International Symposium on Robotics Research*, August 1987, pp. 421–429.

[Wong85] E. K. Wong and K. S. Fu, "A Hierarchical Orthogonal Space Approach to Collision–Free Path Planning," *Proceedings of the 1985 Institute of Electrical and Electronics Engineers International Conference on Robotics and Automation*, March 1985, pp. 506–511.

[Zadeh65] L. A. Zadeh, "Fuzzy Sets," *Information and Control 8*, 3 (June 1965), pp. 338–353.

[Zhu90] D. J. Zhu and J. C. Latombe, "Constraint Reformulation in a Hierarchical Path Planner," *Proceedings of the 1990 Institute of Electrical and Electronics Engineers International Conference of Robotics and Automation*, Cincinnati, 1990, pp. 1918–1923.

[Zweben88] M. Zweben, "Improving Operationality with Approximate Heuristics," *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Explanation–Based Learning*, Palo Alto, CA, March 1988, pp. 100–106.

VITA

Scott William Bennett was born on August 14, 1962, in Iowa City, Iowa. He attended Northwestern University where he received his B.S. degree in Electrical Engineering with honors in 1984. In the fall of 1984, he entered the graduate program in Electrical Engineering at the University of Illinois, and in the Spring joined the explanation-based learning group at the Coordinated Science Laboratory under the direction of Professor Gerald DeJong. He received an M.S. degree in Electrical Engineering in 1989. His thesis demonstrated a technique for learning uncertainty tolerant plans through approximation in complex domains. His work on learning in real-world domains continued at the Beckman Institute where a new learning technique called Permissive Planning was developed for planning in complex, real-world domains. For this work, he was awarded the Ph.D. degree in Electrical Engineering at the University of Illinois at Urbana-Champaign in January 1993. His research interests include machine learning, automated planning, and robotics. He is currently a member of the professional staff of the Systems Research and Applications Corporation in Arlington, Virginia.